Librería de funciones para .Net Framework 2.0/3.0 o superior

SwanCSharp v4.5

Licencia FreeWare

http://www.swancsharp.com
http://www.facebook.com/swancsharp
https://twitter.com/SwanCSharp (@swancsharp)

v4.5 (Enero 2014)

CONTROL DE REVISIONES

VERSIÓN	FECHA	OBSERVACIONES
4.5 Rev1	02/01/2014	Documentación inicial

ÍNDICE DEL DOCUMENTO

1. AGRADECIMIENTOS

Gracias a Pedro Pablo Fernández (http://www.pedropablofernandez.com), por diseñar el logotipo de SwanCSharp.

2. SwanCSharp en Internet

El proyecto SwanCSharp está presente en Internet tanto en la Web oficial del proyecto como en las redes sociales Facebook y Twitter. A continuación se detallan las direcciones de acceso en Internet y los códigos QR para facilitar la conexión mediante telefonía móvil.

2.1 Web Oficial

En la Web oficial de SwanCSharp se puede descargar cualquier versión del proyecto, así como se pueden descargar los manuales en su versión en español o en inglés. También existe un formulario para comunicarse vía email con los responsables del proyecto.

Dirección Web: **http://www.swancsharp.com**

Código QR:

2.2 Facebook

En la red social Facebook existe una página oficial dedicada al proyecto SwanCSharp en el que se pretende crear una comunidad de usuarios de este proyecto.

Dirección Web: **http://www.facebook.com/swancsharp**

Código QR:

2.3 Twitter

En la red social Twitter existe una página oficial dedicada al proyecto SwanCSharp en el que se pretende crear una comunidad de usuarios de este proyecto.

Dirección Web: **https://twitter.com/swancsharp**

Cuenta: @**swancsharp**

Código QR:

3. ACUERDO DE LICENCIA FREEWARE USUARIO FINAL

AVISO AL USUARIO: *Por favor, lea cuidadosamente. Usando todo o una porción del Software está aceptando todos los términos y condiciones de este Acuerdo. Si no está de acuerdo, no use éste Software.*

3.1 DEFINICIONES

Dentro de este Acuerdo, la siguiente terminología tendrá el respectivo significado indicado, tales significados aplicándose tanto a la forma singular y plural de los términos a definir:

"Licenciante" significa http://www.swancsharp.com y Manuel Llaca como desarrollador del producto.

"Concesionario" significa Ud. o Su Compañía, a menos que se indique de otra forma.

"Software" significa (a) todo los contenidos de los archivos, disco(s), CD-ROM(s) o cualquier otro medio del cual este Acuerdo esté previsto, incluyendo pero no limitando a ((i)

información de registro, (ii) material o archivos explicativos relacionados ("Documentación"); y (iii) archivos de configuración, ejecución y ejemplos de código del Software (de haberlos)); y (b) mejoras, versiones modificadas, adiciones y copias del Software, de haberlas, licenciadas a Ud. por http://www.swancsharp.com (colectivamente, "Actualizaciones").

"Uso", "Usar" o "Utilizar" significa acceder, instalar, descargar, copiar o cualquier otro beneficio obtenido de usar las funcionalidades del Software de acuerdo con la Documentación.

"Sistema" significa XO, Windows OS, GNU/Linux o Mac OSX, o cualquier otra máquina virtual.

3.2 DE USO GENERAL

Se le concede a Ud. una Licencia de Uso no exclusiva del Software descargado para cualquier propósito por un período de tiempo ilimitado. El producto de software bajo esta Licencia está provisto gratuitamente. Si bien no se efectúa transacción monetaria por la licencia de uso de este software, eso no significa que no existan condiciones para el uso de dicho software:

- El Software podrá ser instalado y Utilizado por el Concesionario para cualquier propósito legal.

- El Software podrá ser instalado y Utilizado por el Concesionario en cualquier cantidad de Sistemas.

- El Software podrá ser copiado y distribuido bajo la condición de que el copyright y su garantía se mantengan intactas, y el Concesionario no cobre dinero o matrículas por el producto de Software, exceptuando la cobertura de costos de distribución.

- El Concesionario puede referenciar el Software a sus proyectos comerciales, pudiendo cobrar dinero por su software comercial sin tener que abonar regalía o matrícula al Licenciante por distribuir referenciado el Software.

- El Concesionario no tendrá ningún derecho de propiedad de o sobre el Software. El Concesionario reconoce y está de acuerdo de que el Licenciante retiene todos los copyrights y cualquier otro derecho, incluido el de propiedad, de y sobre el Software.

- El Uso en el ámbito de esta Licencia es gratuito y ninguna regalía o matrícula de licencia deberá ser abonada por el Concesionario.

3.3 DERECHOS DE PROPIEDAD INTELECTUAL

- Esta Licencia no transmite ningún derecho intelectual sobre el Software. El Software y cualquiera de las copias que el Concesionario esté autorizado por el Licenciante a realizar son propiedad intelectual del Licenciante y sus proveedores, y le pertenecen.

- El Software está protegido por copyright, incluyendo sin limitación por Leyes de Derecho de Autor y disposiciones de tratados internacionales.

- Cualquier copia que el Concesionario esté permitido a realizar de conformidad con este Acuerdo deberá contener el mismo copyright y cualquier otro anuncio de propiedad que aparezca sobre o en el Software.

- La estructura, organización y código del Software son secreto comercial e información confidencial del Licenciante y sus proveedores. El Concesionario acuerda no descompilar, desmontar o de ninguna otra forma descubrir el código fuente del Software.

- Cualquier intento de ingeniería invertida, copia, clonación, modificación o alteración de cualquier manera al programa instalador sin la aprobación específica del Licenciante están estrictamente prohibidos. El Concesionario no está autorizado para usar ningún plug-in o adherido que permita salvarle modificaciones a un archivo con software licenciado y distribuido por el Licenciante.

- Cualquier información suministrada por el Licenciante u obtenida por el Concesionario, como a continuación permite, podrá ser usada únicamente por el Concesionario con el fin descrito aquí y no podrá divulgarse a ningún tercero o usado para crear ningún software que sea en sustancia similar al expresado por el Software.

- Las Marcas deberán ser utilizadas de acuerdo con la práctica de marcas aceptada, incluyendo identificación de los propietarios de dichas marcas. Las marcas podrán ser utilizadas solamente para identificar cualquier salida impresa del Software y dicho uso de las marcas no otorga al Concesionario ningún tipo de propiedad de y para ellas.

3.4 GARANTÍA

El Licenciante garantiza que:

- Http://www.swancsharp.com poseen propiedad sobre el Software y su documentación y/o está en posesión de los derechos y licencias válidas que apoyan los términos de este Acuerdo.

- Al leal saber y entender del Licenciante, el Software no infringe o viola ningún derecho de propiedad intelectual de ningún tercero.

- El Software no contiene ningún tipo de puerta trasera, bomba de tiempo, dispositivo de autoeliminación o cualquier otra rutina intencionalmente diseñada por el Licenciante para inhabilitar un programa de computadora, o instrucciones que alteren, destruyan o inhiban el entorno de procesamiento.

- Exceptuando aquellas garantías especificadas en la sección 1.4 arriba, el Software está siendo entregado “COMO ES” y el Licenciante no ofrece ninguna garantía en cuanto a su uso o el rendimiento del mismo.

- El Licenciante y sus proveedores no garantizan ni pueden garantizar el rendimiento o resultados que el Concesionario pueda obtener usando el Software. El riesgo que surja del uso o funcionamiento del Software lo asume el Concesionario.

- El Licenciante no da ninguna garantía, expresa o implícita, de que (i) el Software será de calidad satisfactoria, adecuado para algún propósito particular o para cualquier uso particular dentro de condiciones específicas, a pesar de que dicha finalidad, uso o condición pueda ser conocida por el Licenciante; o (ii) que el Software operará libre de errores o sin interrupción o que cualquier error será corregido.

3.5 LIMITACION DE RESPONSABILIDADES

En ninguna circunstancia el Licenciante o sus proveedores serán responsables por cualquier daño, reclamo o costo cualquiera que sea o cualquier daño consecuente, indirecto o incidental, o ninguna pérdida, incluso si el Licenciante ha sido notificado de la posibilidad de dicha pérdida, daño, reclamo o costo realizado por cualquier tercero.

En ninguna circunstancia el Concesionario será responsable para con el Licenciante bajo la condición de que el Concesionario cumpla con todos los términos y condiciones establecidos en esta Licencia.

3.6 CLAUSULAS DE NO-RESCISIÓN

Si alguna porción de este acuerdo se considera inaplicable, el resto se mantendrá válido. Esto significa que si una sección del Acuerdo no es permitida por ley, el resto del Acuerdo sigue en vigor. El no ejercicio de alguno de los derechos bajo este Acuerdo por alguna de las partes, no constituirá una renuncia de (a) ningún otro término o condición de este Acuerdo, o (b) algún derecho en cualquier momento posterior a exigir el cumplimiento exacto y estricto de los términos de este Acuerdo.

4. INTRODUCCIÓN

La librería de funciones SwanCSharp tiene como objetivo recopilar gran cantidad de funciones para aplicar en entornos de desarrollo basados en tecnología .NET Framework, y dichas funciones tienen como objetivos: desarrollar aplicaciones de forma más rápida con eficiencia; poner al alcance de la mano de los programadores principiantes el desarrollo de procesos de elevada complejidad.

La librería de funciones SwanCSharp es de rápida implantación y de sencillo manejo, todo el sistema únicamente consta de un archivo llamado “SwanCSharp.dll”.

5. REQUERIMIENTOS

Para poder desarrollar aplicaciones utilizando las funciones de "SwanCSharp.dll" se necesita un entorno de programación que compile bajo el Framework .NET 2.0 u superior. Las aplicaciones se pueden desarrollar en C# y Visual Basic .Net, utilizando entornos de desarrollo IDE de Microsoft (Visual Studio o versiones Express gratuitas) o utilizar otros entornos de desarrollo como Netbeans, SharpDevelop, o MonoDevelop para Windows, o MonoDevelop para Linux.

Todas las funciones incluidas en "SwanCSharp.dll" son utilizables en cualquier tipo de aplicación a desarrollar según los requerimientos definidos en el párrafo anterior (aplicación de formularios de Windows, Servicio Windows, Servicio Web, Aplicación de Consola, Aplicación de Biblioteca de Clases, etc).

6. Solariem (asistente gráfico para SwanCSharp)

La librería "SwanCSharp" dispone de una aplicación asistente llamada "Solariem" que nos permite generar el código fuente y toda la base de la solución (.SLN) con el acceso a datos, gestión de usuarios, gestión de parámetros, pantalla principal, iconos, etc, listo para ejecutar en Visual Studio 2005 + Wpf Extensions o versiones superiores.

Para más información, visitar la web oficial (http://www.solariem.com). Con Solariem se puede crear la base de una aplicación en C# + SwanCSharp en menos de 10 minutos, en el siguiente enlace se puede ver un vídeo de demostración:

http://www.youtube.com/watch?v=V3dw9cbZShw

7. DESCARGA E INSTALACIÓN

La librería "SwanCSharp" se puede descargar de la Web oficial del proyecto (http://www.swancsharp.com), obteniendo un archivo ZIP llamado "SwanCSharp.zip". Se descomprime el archivo ZIP en el lugar deseado del disco duro.

Una vez descomprimido el archivo, en la carpeta "**Binaries**" se encuentra "**SwanCSharp.dll**", archivo que se debe de copiar al directorio de compilación del proyecto de Visual Basic o C# en el que deseemos utilizar las funciones de la librería. En el entorno de desarrollo (IDE) habitual de trabajo se crea un nuevo proyecto, o se abre un proyecto ya existente, y se incluye "SwanCSharp.dll" en la lista de referencias. A partir de ese instante se puede utilizar en dicho proyecto todas las funciones de la librería.

En la carpeta "**Documentation**" se pueden encontrar el manual de uso en castellano y en inglés de la librería.

En la carpeta "**Samples**" se pueden encontrar múltiples ejemplos de utilización de las funciones incorporadas a la librería. Todos los ejemplos han sido desarrollados en el entorno IDE Microsoft Visual C#.

8. NOVEDADES EN LA VERSION 4.5

En este apartado se comentan las novedades que incorpora “SwanCSharp” en esta nueva versión 4.5 con respecto a la versión 4.0 anterior.

8.1 CheckInternetConnection

En la clase “Internet” se añade la función “CheckInternetConnection” que devuelve si hay o no conexión a Internet.

8.2 CheckTCPPortIsOpen

En la clase “Network” se añade la función “CheckTCPPortIsOpen” que dada una dirección IP y un puerto TCP concreto, devuelve si está abierto o no.

8.3 ClipboardAgent

En el espacio de nombres “SwanCSharp_Controls” se crea una clase con un agente del portapapeles de Windows para usar en ventanas “WindowBase”.

8.4 GetDomainNameFromIP

En la clase “Internet” se añade la función “GetDomainNameFromIP” que devuelve el nombre de dominio de una IP válida dada.

8.5 GetFileFromHttp

En la clase “Internet” se añade la función “GetFileFromHttp” que nos permite descargar un archivo subido a Internet.

8.6 GetIPFromDomainName

En la clase “Internet” se añade la función “GetIPFromDomainName” que devuelve la IP de un nombre de dominio de internet dado.

8.7 GetIPNetworkData

En la clase “Network” se añade la función “GetIPNetworkData” que devuelve toda la información asociada a las tarjetas de red Ethernet del sistema.

8.8 GetListOfCountries

En la clase “Miscellaneous” se añade la función “GetListOfCountries” que devuelve una lista de países (en inglés o español).

8.9 GetWebProxyActive

En la clase “Internet” se añade la función “GetWebProxyActive” que nos permite obtener toda la información del proxy web activo en el equipo de ejecución.

8.10 GlobalHotKeys

En el espacio de nombres “SwanCSharp_Controls” se crea una clase nos permite establecer teclas de acceso rápido globales en ventanas “WindowBase”.

8.11 IsLocalIP

En la clase “Network” se añade la función “IsLocalIP” que comprueba si una IP dada es de rango local o no.

8.12 IsValidIP

En la clase “Network” se añade la función “IsValidIP” que comprueba si una IP dada tiene la estructura correcta.

8.13 MySQL

A la gestión de bases de datos en SwanCSharp se añade soporte para el gestor MySQL, ahora se soportan Access, SQL Server, Oracle, Firebird, y MySQL.

8.14 Network

Se crea una clase nueva llamada “Network” que recopila funciones y métodos para la gestión de redes informáticas.

8.15 ShowNotifyButton

En el interfaz gráfico “SwanCSharp_Controls” se crea un nuevo botón de notificación que acompaña a los botones de minimizar, maximizar, y cerrar.

8.16 WindowRemoveUser

Se repara un error al validar los permisos del usuario activo para eliminar una cuenta determinada.

9. ESTRUCTURA DE LA LIBRERIA

La librería “SwanCSharp” se compone de un único archivo dll (**SwanCSharp.dll**) que contiene varios espacios de nombres.

9.1 Espacio de nombres SwanCSharp

El espacio de nombres "**SwanCSharp**" se divide en grupos por clases que son los siguientes (**se puede utilizar este espacio de nombres con el .Net Framework 2.0 o superior instalado**):

SwanCSharp.Arrays.- Clase que incorpora funciones y métodos para gestionar Arrays, evitando así las carencias de dicha clase en C#.

SwanCSharp.Configurator.- Clase que incorpora a nuestros desarrollos una gestión completa de parámetros de configuración (incluye formularios).

SwanCSharp.CRC32.- Clase que incorpora a nuestros desarrollos funciones de cálculo de CRC32 sobre cadenas y ficheros.

SwanCSharp.DataAccess.- Clase que nos permite incorporar a nuestros desarrollos una gestión completa de Acceso a Datos, incluyendo gestión en la creación y mantenimiento de las BBDD.

SwanCSharp.Directories.- Clase que contiene funciones para la gestión y manejo de directorios de carpetas.

SwanCSharp.Encryption.- Clase que contiene funciones para el cifrado de datos suelos o ficheros de cualquier tipo (Cifrado AES256).

SwanCSharp.Files.- Clase que contiene funciones para gestión y mantenimiento de ficheros.

SwanCSharp.Imaging.- Clase que contiene funciones para la gestión y tratamiento de imágenes.

SwanCSharp.Internet.- Clase que contiene funciones relacionadas con el mundo de Internet (FTP Client, etc.).

SwanCSharp.Logger.- Clase que nos permite incorporar a nuestros desarrollos una gestión de archivos de Log de control de excepciones.

SwanCSharp.Miscellaneous.- Clase genérica donde se incluyen aquellas funciones de uso y ámbito general.

SwanCSharp.Network.- Clase que contiene funciones relacionadas con la gestión de redes informáticas.

SwanCSharp.Reporting.- Clase que incorpora funciones y métodos para desarrollar fácilmente y rápidamente informes de datos en HTML para su posterior visualización o impresión mediante un visualizador integrado.

SwanCSharp.SNTP.- Clase que incorpora funciones y métodos para incorporar a nuestros desarrollos un cliente de tiempo SNTP.

SwanCSharp.Sockets.- Clase que incorpora funciones y métodos parar desarrollar fácilmente y rápidamente un Servidor Socket o un Cliente Socket, o ambos; permitiendo realizar aplicaciones que habilitan una comunicación cliente/servidor fluida utilizando cualquier conexión (LAN, WAN o Internet) mediante el puerto TCP deseado.

SwanCSharp.UserManagement- Clase que incorpora a nuestros desarrollos una gestión completa de usuarios y perfiles (incluye formularios). Hay una clase específica para cada gestor de bases de datos (SQLServer, Oracle, Firebird, y Access).

SwanCSharp.Validations.- Clase que permite incorporar a nuestros desarrollos funciones para validar datos.

SwanCSharp.Video.- Clase que permite gestionar streaming de video generado por cámaras de video IP.

9.2 Espacio de nombres SwanCSharp_Controls

El espacio de nombres "**SwanCSharp_Controls**" se divide en grupos por clases que son los siguientes (**se puede utilizar este espacio de nombres con el .Net Framework 3.0 o superior instalado**):

SwanCSharp_Controls.ClipboardAgent.- Se crea esta clase para incluir un agente que "vigila" por el movimiento de contenidos en el portapapeles de Windows.

SwanCSharp_Controls.GlobalHotKeys.- Se crea esta clase para incluir gestión global de teclas de acceso rápido.

SwanCSharp_Controls.SW_Miscellaneous.- Se crea esta clase genérica donde se incluyen aquellas funciones existentes en SwanCSharp.Miscellaneous que requieren de una personalización para el uso de SwanCSharp.WindowBase.

SwanCSharp_Controls.WindowBase.- Clase que incorpora el diseño básico de todas las ventanas que se pueden crear con esta librería.

SwanCSharp_Controls.WindowAddUser.- Clase que incorpora el diseño básico de la ventana de creación de usuarios.

SwanCSharp_Controls.WindowDataQuery.- Clase que incorpora el diseño básico de la ventana de visualización de datos.

SwanCSharp_Controls.WindowError.- Clase que incorpora el diseño básico de la ventana de mensajes de error.

SwanCSharp_Controls.WindowFile.- Clase que incorpora el diseño básico de la ventana de selección de archivos (cuadro de diálogo para abrir archivos).

SwanCSharp_Controls.WindowFolder.- Clase que incorpora el diseño básico de la ventana de selección de carpetas.

SwanCSharp_Controls.WindowInformation.- Clase que incorpora el diseño básico de la ventana de mensajes de información.

SwanCSharp_Controls.WindowLoginUser.- Clase que incorpora el diseño básico de la ventana de acceso a usuarios.

SwanCSharp_Controls.WindowNewFile.- Clase que incorpora el diseño básico de la ventana de selección de archivos (cuadro de diálogo para nuevos archivos).

SwanCSharp_Controls.WindowOKCancelQuestion.- Clase que incorpora el diseño básico de la ventana de preguntas.

SwanCSharp_Controls.WindowParameters.- Clase que incorpora el diseño básico de la ventana de parámetros de configuración.

SwanCSharp_Controls.WindowPasswordUser.- Clase que incorpora el diseño básico de la ventana de cambio de contraseñas de usuarios.

SwanCSharp_Controls.WindowRemoveUser.- Clase que incorpora el diseño básico de la ventana de borrado de usuarios.

SwanCSharp_Controls.WindowReportView.- Clase que incorpora el diseño básico de la ventana de visualización de informes.

SwanCSharp_Controls.WindowSplash.- Clase que incorpora el diseño básico de la ventana de bienvenida.

SwanCSharp_Controls.WindowWarning.- Clase que incorpora el diseño básico de la ventana de mensajes de aviso.

SwanCSharp_Controls.WindowYesNoQuestion.- Clase que incorpora el diseño básico de la ventana de preguntas.

10. REFERENCIA DE USO DE LAS FUNCIONES (SwanCSharp)

A continuación se va a describir la referencia de uso de cada una de las funciones incorporadas a la librería dentro del espacio de nombres SwanCSharp. Las referencias se agrupan por las clases a las que pertenecen en orden alfabético.

10.1 SwanCSharp.Arrays

La clase Arrays incorpora una serie de funciones de asistencia con el manejo de arrays de diferentes tipos. Esta clase es muy útil para redimensionado de ese tipo de objetos.

10.1.1 AddByteToArray

La función permite añadir un byte al final o al principio de un array de bytes determinado, redimensionando el array original. Espera recibir tres parámetros: en el primer parámetro el array origen, en el segundo parámetro el byte a agregar, y en el tercer parámetro un valor booleano, verdadero lo inserta al principio del array, falso lo inserta al final del array.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
byte[] lobjArraySource = new byte[3];

byte pobjByte1 = 48; // Byte or ASCII code for '0'
byte pobjByte2 = 49; // Byte or ASCII code for '1'
byte pobjByte3 = 50; // Byte or ASCII code for '2'

lobjArraySource[0] = pobjByte1;
lobjArraySource[1] = pobjByte2;
lobjArraySource[2] = pobjByte3;

lobjArraySource = Arrays.AddByteToArray(lobjArraySource, 52, false);
lobjArraySource = Arrays.AddByteToArray(lobjArraySource, 51, true);
```

10.1.2 AddElementsByteArray

La función permite añadir a un array de bytes origen todo el contenido de otro array de bytes secundario. Espera recibir tres parámetros: en el primer parámetro el array origen, en el segundo parámetro el array secundario, y en el tercer parámetro un valor booleano, verdadero lo inserta al principio del array, falso lo inserta al final del array.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
byte[] lobjArraySource = new byte[3];
byte[] lobjArraySecondary = new byte[3];

byte pobjByte1 = 48; // Byte or ASCII code for '0'
byte pobjByte2 = 49; // Byte or ASCII code for '1'
byte pobjByte3 = 50; // Byte or ASCII code for '2'

byte pobjByte4 = 51; // Byte or ASCII code for '3'
byte pobjByte5 = 52; // Byte or ASCII code for '4'
byte pobjByte6 = 53; // Byte or ASCII code for '5'

lobjArraySource[0] = pobjByte1;
lobjArraySource[1] = pobjByte2;
lobjArraySource[2] = pobjByte3;

lobjArraySecondary [0] = pobjByte4;
lobjArraySecondary [1] = pobjByte5;
lobjArraySecondary [2] = pobjByte6;

lobjArraySource = Arrays.AddElementsByteArray(lobjArraySource,
lobjArraySecondary, false);
```

10.1.3 AddElementsIntegerArray

La función permite añadir a un array de enteros origen todo el contenido de otro array de enteros secundario. Espera recibir tres parámetros: en el primer parámetro el array origen, en el segundo parámetro el array secundario, y en el tercer parámetro un valor booleano, verdadero lo inserta al principio del array, falso lo inserta al final del array.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
int[] lobjArraySource = new int[3];
int[] lobjArraySecondary = new int[3];
```

```
int pobjByte1 = 48; // Byte or ASCII code for '0'
int pobjByte2 = 49; // Byte or ASCII code for '1'
int pobjByte3 = 50; // Byte or ASCII code for '2'

int pobjByte4 = 51; // Byte or ASCII code for '3'
int pobjByte5 = 52; // Byte or ASCII code for '4'
int pobjByte6 = 53; // Byte or ASCII code for '5'

lobjArraySource[0] = pobjByte1;
lobjArraySource[1] = pobjByte2;
lobjArraySource[2] = pobjByte3;

lobjArraySecondary [0] = pobjByte4;
lobjArraySecondary [1] = pobjByte5;
lobjArraySecondary [2] = pobjByte6;

lobjArraySource = Arrays.AddElementsIntegerArray(lobjArraySource,
lobjArraySecondary, false);
```

10.1.4 AddElementsObjectArray

La función permite añadir a un array de objetos origen todo el contenido de otro array de objetos secundario. Espera recibir tres parámetros: en el primer parámetro el array origen, en el segundo parámetro el array secundario, y en el tercer parámetro un valor booleano, verdadero lo inserta al principio del array, falso lo inserta al final del array.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
object[] lobjArraySource = new object[3];
object[] lobjArraySecondary = new object[3];

object pobjByte1 = 48; // Byte or ASCII code for '0'
object pobjByte2 = 49; // Byte or ASCII code for '1'
object pobjByte3 = 50; // Byte or ASCII code for '2'

object pobjByte4 = 51; // Byte or ASCII code for '3'
object pobjByte5 = 52; // Byte or ASCII code for '4'
object pobjByte6 = 53; // Byte or ASCII code for '5'

lobjArraySource[0] = pobjByte1;
lobjArraySource[1] = pobjByte2;
lobjArraySource[2] = pobjByte3;

lobjArraySecondary [0] = pobjByte4;
lobjArraySecondary [1] = pobjByte5;
lobjArraySecondary [2] = pobjByte6;

lobjArraySource = Arrays.AddElementsObjectArray(lobjArraySource,
lobjArraySecondary, false);
```

10.1.5 AddElementsStringArray

La función permite añadir a un array de strings origen todo el contenido de otro array de strings secundario. Espera recibir tres parámetros: en el primer parámetro el array origen, en el segundo parámetro el array secundario, y en el tercer parámetro un valor booleano, verdadero lo inserta al principio del array, falso lo inserta al final del array.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
string[] lobjArraySource = new string[3];
string[] lobjArraySecondary = new string[3];

string pobjByte1 = “48”; // Byte or ASCII code for '0'
string pobjByte2 = “49”; // Byte or ASCII code for '1'
string pobjByte3 = “50”; // Byte or ASCII code for '2'

string pobjByte4 = “51”; // Byte or ASCII code for '3'
string pobjByte5 = “52”; // Byte or ASCII code for '4'
string pobjByte6 = “53”; // Byte or ASCII code for '5'

lobjArraySource[0] = pobjByte1;
lobjArraySource[1] = pobjByte2;
lobjArraySource[2] = pobjByte3;

lobjArraySecondary [0] = pobjByte4;
lobjArraySecondary [1] = pobjByte5;
lobjArraySecondary [2] = pobjByte6;

lobjArraySource = Arrays.AddElementsStringArray(lobjArraySource,
lobjArraySecondary, false);
```

10.1.6 AddIntegerToArray

La función permite añadir un entero al final o al principio de un array de enteros determinado, redimensionando el array original. Espera recibir tres parámetros: en el primer parámetro el array origen, en el segundo parámetro el entero a agregar, y en el tercer parámetro un valor booleano, verdadero lo inserta al principio del array, falso lo inserta al final del array.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
int[] lobjArraySource = new int[3];
```

```
int pobjByte1 = 48; // Byte or ASCII code for '0'
int pobjByte2 = 49; // Byte or ASCII code for '1'
int pobjByte3 = 50; // Byte or ASCII code for '2'

lobjArraySource[0] = pobjByte1;
lobjArraySource[1] = pobjByte2;
lobjArraySource[2] = pobjByte3;

lobjArraySource = Arrays.AddIntegerToArray(lobjArraySource, 52, false);
lobjArraySource = Arrays.AddIntegerToArray(lobjArraySource, 51, true);
```

10.1.7 AddObjectToArray

La función permite añadir un objeto al final o al principio de un array de objetos determinado, redimensionando el array original. Espera recibir tres parámetros: en el primer parámetro el array origen, en el segundo parámetro el objeto a agregar, y en el tercer parámetro un valor booleano, verdadero lo inserta al principio del array, falso lo inserta al final del array.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
object[] lobjArraySource = new object[3];

object pobjByte1 = 48; // Byte or ASCII code for '0'
object pobjByte2 = 49; // Byte or ASCII code for '1'
object pobjByte3 = 50; // Byte or ASCII code for '2'

lobjArraySource[0] = pobjByte1;
lobjArraySource[1] = pobjByte2;
lobjArraySource[2] = pobjByte3;

lobjArraySource = Arrays.AddObjectToArray(lobjArraySource, 52, false);
lobjArraySource = Arrays.AddObjectToArray(lobjArraySource, 51, true);
```

10.1.8 AddStringToArray

La función permite añadir un string al final o al principio de un array de strings determinado, redimensionando el array original. Espera recibir tres parámetros: en el primer parámetro el array origen, en el segundo parámetro el string a agregar, y en el tercer parámetro un valor booleano, verdadero lo inserta al principio del array, falso lo inserta al final del array.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
string[] lobjArraySource = new string[3];

string pobjByte1 = “48”; // Byte or ASCII code for '0'
string pobjByte2 = “49”; // Byte or ASCII code for '1'
string pobjByte3 = “50”; // Byte or ASCII code for '2'

lobjArraySource[0] = pobjByte1;
lobjArraySource[1] = pobjByte2;
lobjArraySource[2] = pobjByte3;

lobjArraySource = Arrays.AddStringToArray(lobjArraySource, “52”, false);
lobjArraySource = Arrays.AddStringToArray(lobjArraySource, “51”, true);
```

10.1.9 ArrayResize

La función “ArrayResize” tiene como objetivo redimensionar variables de tipo Array mediante una única función más genérica.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
string lstrArray = (string[])Arrays.ArrayResize(pstrArray, pstrArray.Length
+ 1);
```

10.1.10 FindItemInByteArray

La función permite buscar un byte dentro de un array de bytes. Devuelve verdadero si lo encuentra y devuelve falso si no lo encuentra.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
Boolean lblnExists = Array.FindItemInByteArray(lobjByte,larrBytes);
```

10.1.11 FindItemInIntegerArray

La función permite buscar un entero dentro de un array de enteros. Devuelve verdadero si lo encuentra y devuelve falso si no lo encuentra.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
Boolean lblnExists =
Array.FindItemInIntegerArray(lintInteger,larrIntegers);
```

10.1.12 FindItemInObjectArray

La función permite buscar un objeto dentro de un array de objetos. Devuelve verdadero si lo encuentra y devuelve falso si no lo encuentra.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
Boolean lblnExists = Array.FindItemInObjectArray(lobjObject,larrObjects);
```

10.1.13 FindItemInStringArray

La función permite buscar un string dentro de un array de strings. Devuelve verdadero si lo encuentra y devuelve falso si no lo encuentra.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
Boolean lblnExists = Array.FindItemInStringArray(lstrString,larrStrings);
```

10.1.14 RemoveItemsFromByteArray

La función permite eliminar un número secuencial de elementos de un array de bytes. Espera recibir tres parámetros: en el primer parámetro el array origen, en el segundo parámetro el elemento índice de inicio que se desea borrar, y en el tercer parámetro el elemento índice final de borrado. La función devolverá un array de bytes con los elementos borrados que van desde el inicio hasta el final.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
byte[] lobjArraySource = new byte[3];

byte pobjByte1 = 48;
byte pobjByte2 = 49;
byte pobjByte3 = 50;

lobjArraySource[0] = pobjByte1;
lobjArraySource[1] = pobjByte2;
lobjArraySource[2] = pobjByte3;

lobjArraySource = Array.RemoveItemsFromByteArray(lobjArraySource,1,2);
```

10.1.15 RemoveItemsFromIntegerArray

La función permite eliminar un número secuencial de elementos de un array de enteros. Espera recibir tres parámetros: en el primer parámetro el array origen, en el segundo parámetro el elemento índice de inicio que se desea borrar, y en el tercer parámetro el elemento índice final de borrado. La función devolverá un array de enteros con los elementos borrados que van desde el inicio hasta el final.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
int[] lobjArraySource = new int[3];

int pobjByte1 = 48;
int pobjByte2 = 49;
int pobjByte3 = 50;

lobjArraySource[0] = pobjByte1;
lobjArraySource[1] = pobjByte2;
lobjArraySource[2] = pobjByte3;

lobjArraySource = Array.RemoveItemsFromIntegerArray(lobjArraySource,1,2);
```

10.1.16 RemoveItemsFromObjectArray

La función permite eliminar un número secuencial de elementos de un array de objetos. Espera recibir tres parámetros: en el primer parámetro el array origen, en el segundo parámetro el elemento índice de inicio que se desea borrar, y en el tercer parámetro el elemento índice final de borrado. La función devolverá un array de objetos con los elementos borrados que van desde el inicio hasta el final.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
object[] lobjArraySource = new object[3];

object pobjByte1 = 48;
object pobjByte2 = 49;
object pobjByte3 = 50;

lobjArraySource[0] = pobjByte1;
lobjArraySource[1] = pobjByte2;
lobjArraySource[2] = pobjByte3;

lobjArraySource = Array.RemoveItemsFromObjectArray(lobjArraySource,1,2);
```

10.1.17 RemoveItemsFromStringArray

La función permite eliminar un número secuencial de elementos de un array de strings. Espera recibir tres parámetros: en el primer parámetro el array origen, en el segundo parámetro el elemento índice de inicio que se desea borrar, y en el tercer parámetro el elemento índice final de borrado. La función devolverá un array de strings con los elementos borrados que van desde el inicio hasta el final.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
string[] lobjArraySource = new string[3];

string pobjByte1 = “48”;
string pobjByte2 = “49”;
string pobjByte3 = “50”;

lobjArraySource[0] = pobjByte1;
lobjArraySource[1] = pobjByte2;
lobjArraySource[2] = pobjByte3;

lobjArraySource = Array.RemoveItemsFromStringArray(lobjArraySource,1,2);
```

10.2 SwanCSharp.Configurator

La clase "Configurator” nos permite crear un sistema completo de gestión de parámetros de configuración en las aplicaciones desarrolladas.

10.2.1 Configurator

El constructor de la clase “Configurator” requiere un soporte para almacenar los datos de configuración, pudiendo seleccionar entre SQL Server, Oracle, Firebird, Access, MySQL, o un fichero de texto estándar. Existe una clase específica para cada gestor de base de datos (ConfiguratorSQLServer, ConfiguratorOracle, ConfiguratorFirebird, ConfiguratorAccess, ConfiguratorMySQL, y ConfiguratorFile). La finalidad de estas clases es la de incorporar a una aplicación desarrollada una gestión completa de los parámetros de configuración (incluidas ventanas de gestión). Por eso, cada vez que se construya el objeto “Configurator”, esta clase comprobará si en nuestra base de datos existe la tabla “AppConfig” y si dispone de los campos necesarios. En caso contrario lo primero que hará de forma automática es crear toda la estructura necesaria en nuestra base de datos elegida (proceso que únicamente ejecutará la primera vez y sin intervención del desarrollador). En el caso de que la opción elegida para alojar la configuración sea un fichero grabado en disco, el constructor creará una carpeta "Config" colgando del directorio de ejecución de la aplicación y allí creará el archivo de configuración (pasado por parámetro en el constructor).

En el caso que el soporte de datos elegido sea SQL Server, los datos de la tabla “AppConfig” serán cifrados por la clase “ConfiguratorSQLServer” para que nadie externamente a nuestros desarrollos pueda acceder a los. Por lo tanto esta clase utiliza cifrado internamente para mayor seguridad.

En el caso de que la base de datos elegida sea Access, los datos no se cifran, pero se recomienda al usuario que proteja la base de datos mediante contraseña (desde la opción Seguridad de Microsoft Access).

En el caso de que la opción de grabación de la configuración sea un archivo, el contenido será cifrado.

En el resto de los casos (Oracle, Firebird, y MySQL) los datos no se cifran, por lo tanto se recomienda usar los métodos de autenticación que incluyen los propios gestores de bases de datos.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

También es importante informar que las clases “Configurator” se utilizarán a lo largo de todo el desarrollo de nuestra aplicación y por eso se recomienda crear el objeto mediante una variable global para que los datos de configuración, y las llamadas a las funciones de gestión, estén accesibles desde cualquier lugar de la aplicación.

Para crear el configurador sobre SQL Server:

```
public ConfiguratorSQLServer gobjConfig;

private void Form1_Load(object sender, EventArgs e)
{
```

```
  gobjConfig = new ConfiguratorSQLServer("COMPUTER\\INSTANCE", "DataBase");
}
```

Para crear el configurador sobre Oracle:

```
public ConfiguratorOracle gobjConfig;

private void Form1_Load(object sender, EventArgs e)
{
  gobjConfig = new ConfiguratorOracle("Data Source", "User SQL", "Password
SQL" );
}
```

Para crear el configurador sobre Firebird:

```
public ConfiguratorFirebird gobjConfig;

private void Form1_Load(object sender, EventArgs e)
{
  gobjConfig = new ConfiguratorFirebird("Data Source", "File database with
full path", "User SQL", "Password SQL", TCP Port );
}
```

Para crear el configurador sobre MySQL:

```
public ConfiguratorMySQL gobjConfig;

private void Form1_Load(object sender, EventArgs e)
{
  gobjConfig = new ConfiguratorMySQL("Server address", "File database with
full path", "User SQL", "Password SQL", TCP Port );
}
```

Para crear el configurador sobre Access:

```
public ConfiguratorAccess gobjConfig;

private void Form1_Load(object sender, EventArgs e)
{
  gobjConfig = new ConfiguratorAccess("database filename", "path database",
"password");
}
```

Para crear el configurador sobre un fichero grabado en disco:

```
public ConfiguratorFile gobjConfig;

private void Form1_Load(object sender, EventArgs e)
{
```

```
  gobjConfig = new ConfiguratorFile("filename.cfg");
}
```

10.2.1.1 AddNewConfigParameter

Se utiliza para crear un nuevo parámetro de configuración. Necesita tres parámetros, uno para indicar qué nombre va a tener el nuevo parámetro; el segundo para indicar el valor de dicho parámetro; el tercero para indicar si el parámetro va a ser visible para el usuario o solo para el uso interno de la aplicación. Los dos primeros parámetros son de tipo "String". Por ejemplo:

```
gobjConfig.AddNewConfigParameter("PathFiles", "c:\\test", true);
gobjConfig.AddNewConfigParameter("TimeOut", "15", false);
```

10.2.1.2 ExistConfigParameter

Se utiliza para comprobar si un parámetro existe en la configuración actual. Solo necesita un parámetro que se corresponde con el nombre del parámetro a buscar. Por ejemplo:

```
if (gobjConfig.ExistsConfigParameter("PathFiles"))
{
}
```

10.2.1.3 GetConfigParameterValue

Se utiliza para recuperar el valor de un parámetro existente. Solo necesita un parámetro que se corresponde con el nombre del parámetro a buscar. Por ejemplo:

```
if (gobjConfig.ExistsConfigParameter("PathFiles"))
{
    string lstrPathFiles = gobjConfig.GetConfigParameterValue("PathFiles");
}
```

10.2.1.4 GetDataTableFromAppConfig

Se utiliza para obtener un objeto DataTable con todos los datos relacionados con los parámetros. Se recibe un DataTable que contiene los campos IDParameter, CF_Name_Parameter, CF_Value_Parameter, y CF_Visible_Parameter. Un ejemplo de uso de la función es:

```
private DataTable ldatParameters;
ldatParameters = gobjConfigurator.GetDataTableFromAppConfig();
```

10.2.1.5 GetDataTableFromAppConfigOnlyVisible

Se utiliza para obtener un objeto DataTable con todos los datos relacionados con los parámetros, pero solo se obtienen los marcados como "Visibles". Se recibe un DataTable que contiene los campos IDParameter, CF_Name_Parameter, CF_Value_Parameter, y CF_Visible_Parameter. Un ejemplo de uso de la función es:

```
private DataTable ldatParameters;
ldatParameters = gobjConfigurator.GetDataTableFromAppConfigOnlyVisible();
```

10.2.1.6 RemoveConfigParameter

Se utiliza para eliminar un parámetro existente. Solo necesita un parámetro que se corresponde con el nombre del parámetro a eliminar. Por ejemplo:

```
gobjConfig.RemoveConfigParameter("PathFiles");
```

10.2.1.7 ShowConfigurationWindow

La ventana de "Configuration Window" nos permitirá visualizar y modificar en pantalla todos los parámetros de configuración asociados a nuestros desarrollos. La ventana permite mostrarse en dos modos diferentes; un modo para el administrador donde podrá añadir, modificar, y eliminar todos los parámetros (visibles y no visibles); otro modo donde se podrán consultar y modificar únicamente los parámetros visibles, no pudiendo en ningún caso crear uno nuevo o eliminar un parámetro de configuración.

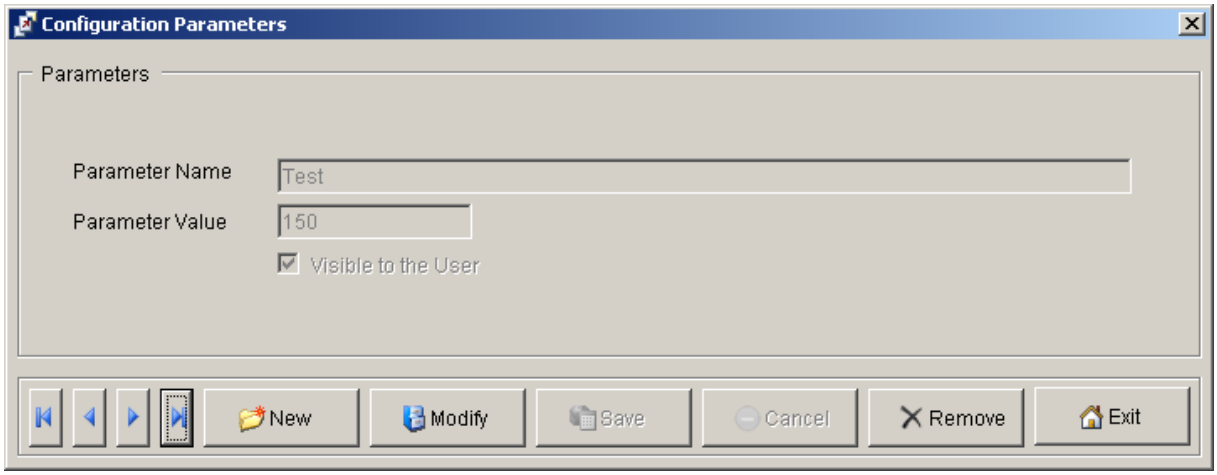
A modo de sugerencia podemos comentar que la clase de configuración se puede combinar con la clase de gestión de usuarios de SwanCSharp, de tal forma que podemos crear dos opciones en un MenuStrip, una opción que únicamente se habilita para los usuarios de perfiles Superusuario (desarrollador de la aplicación) para que acceda con todos los permisos a la configuración de parámetros. Otra opción que únicamente se habilita para administradores (usuarios con privilegios de configuración), que podrán modificar los valores de aquellos parámetros "visibles". El tercer perfil de usuarios simples no tendrá acceso a la ventana parámetros.

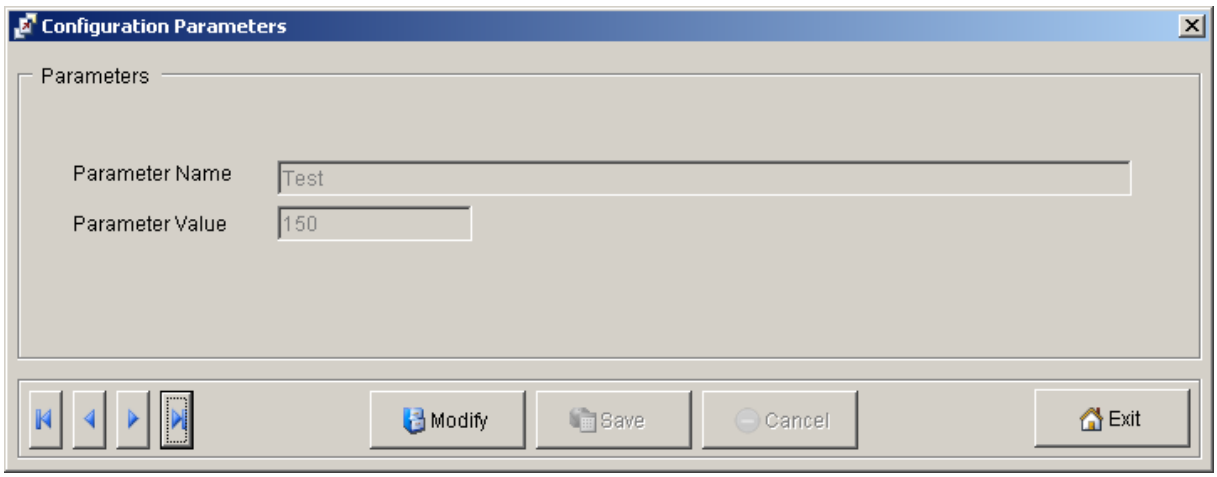
Para llamar a la ventana de modificación de parámetros con todos los permisos podemos escribir el siguiente código:

```
gobjConfig.ShowConfigurationWindow(false, true, InterfaceLanguage.English);
```

Para llamar a la ventana de modificación de parámetros solo para visualizar y modificar los parámetros "visibles" podemos escribir el siguiente código:

```
gobjConfig.ShowConfigurationWindow(true, false, InterfaceLanguage.English);
```





Nota importante: En el caso de los formularios personalizados “SwanCSharp” creados desde el espacio de nombres “SwanCSharp_Controls”, será necesario utilizar la clase “WindowParameters” existente en ese espacio de nombres en lugar de esta función “ShowConfigurationWindow”.

10.2.1.8 UpdateConfigParameterValue

Se utiliza para actualizar el valor de un parámetro existente. Necesita tres parámetros, el primero para indicar el nombre del parámetro a buscar, el segundo para especificar el nuevo valor de dicho parámetro, y el tercero para especificar si el parámetro será visible para el usuario o únicamente de uso interno de la aplicación. Por ejemplo:

```
gobjConfig.UpdateConfigParameterValue("PathFiles", "c:\\test2", true);
```

10.3 SwanCSharp.CRC32

La clase "CRC32" nos permite calcular el valor de CRC (32 bits) de cualquier fichero especificado, o también puede calcular el CRC de una cadena de texto determinada (existen 2 sobrecargas). Esta función es útil para comprobar si un fichero ha cambiado en su

estructura, por ejemplo, se puede utilizar para comprobar que un fichero original ejecutable (EXE) no ha sido modificado en ningún momento, y así proteger de cualquier cambio malintencionado.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
string lstrHash = "";
CRC32 lobjCRC = new CRC32("", "Pruebas.exe");
lstrHash = lobjCRC.Hash;
```

El constructor de la clase "CRC32" espera recibir en el primer parámetro la ruta del fichero, y en el segundo parámetro el nombre del fichero.

10.4 SwanCSharp.DataAccess

La clase DataAccess permite gestionar desde código fuente bases de datos de una forma eficiente y rápida, soportando las bases de datos SQL Server, Oracle, Firebird, MySQL, y Access, existiendo una clase DataConnection para cada uno de los gestores de datos (DatabaseConnectionSQLServer, DatabaseConnectionOracle, DatabaseConnectionFirebird, DatabaseConnectionMySQL, DatabaseConnectionAccess). Las clases funcionan mediante la implementación de instrucciones SQL estándar, y dispone de una función para crear la conexión con la base de datos, otra función para cerrar la conexión, y dos funciones de ejecución de comandos, una para ejecutar comandos SELECT, y otra para ejecutar comandos INSERT, UPDATE, DELETE, o comandos restantes.

También se dispone de cuatro clases secundarias (DatabaseStructureSQLServer, DatabaseStructureOracle, DatabaseStructureFirebird, DatabaseStructureMySQL, y DatabaseStructureAccess) que permite crear y actualizar bases de datos de SQL Server, Oracle, Firebird, MySQL, o Access en tiempo de ejecución dinámicamente, permitiendo crear aplicaciones que crean y actualizan automáticamente las bases de datos.

10.4.1 DataConnectionSQLServer, DataConnectionOracle, DataConnectionFirebird, DataConnectionMySQL y DataConnectionAccess

Existen cuatro clases de conexión (SQLServer, Oracle, Firebird, MySQL, y Access). Por lo tanto las cuatro clases DataAccess se pueden utilizar para realizar conexiones más sencillas y portables localmente o remotamente.

Para poder utilizar la clase DataConnectionSQLServer es necesario tener acceso en el sistema a una versión “Comercial” o “Express” (gratuita) de SQL Server, ya sea con acceso local (instalada en el mismo equipo) o acceso remoto (instalado en otro equipo).

Para poder utilizar la clase DataConnectionOracle es necesario tener acceso en el sistema una versión “Comercial” o “Express” (gratuita) de Oracle, ya sea con acceso local (instalada en el mismo equipo) o acceso remoto (instalado en otro equipo). En el equipo donde va a ser ejecutada la aplicación que usa DataConnectionOracle es necesario que se instale el software de Oracle de conexión para .Net llamado ODP .Net (una versión válida para el motor de base de datos Oracle que esté instalado en el sistema.

Para poder utilizar la clase DataConnectionFirebird es necesario tener acceso en el sistema una versión de Firebird, además de copiar junto con “SwanCSharp.dll” el archivo “.NET Provider” para Firebird (se recomienda el archivo “FirebirdSql.Data.FirebirdClient.dll” en la versión 2.5.2).

Para poder utilizar la clase DataConnectionMySQL es necesario tener instalado previamente en el sistema el software de conexión gratuito “MySQL Connector/Net” de la versión 6.7.4 o superior (se puede descargar en http://dev.mysql.com/downloads/connector/net/).

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.DataAccess;
```

Para crear la conexión con una base de datos de Microsoft SQL Server se ejecuta:

```
DataConnectionSQLServer lobjConnection = new DataConnectionSQLServer("PC-
NAME\\INSTANCE SQL", "DataBase");
```

En el primer parámetro se pasa el nombre de la instancia de SQL Server instalada indicando también el nombre del PC en Windows. En el segundo parámetro se indica el nombre de la base de datos creada en SQL Server, en el tercer parámetro se pasa un enumerado especificando que la base de datos es de Microsoft SQL Server. Si el Microsoft SQL Server no está instalado localmente, y está instalado remotamente por medio de una dirección LAN o WAN accesible desde el equipo donde se va a ejecutar el desarrollo, en el primer parámetro instancia se pasaría “IP ServerSQL\\INSTANCE SQL”. Pero para poder realizar conexiones remotas desde la clase SwanCSharp.DataAccess, primero es necesario haber habilitado conexiones remotas en el Microsoft SQL Server instalado en el equipo remoto, y haber dado todos los permisos necesarios. Existe otra sobrecarga que permite, además, pasar el usuario y contraseña para la autenticación SQL en lugar de la autenticación Windows.

Para crear la conexión con una base de datos de Oracle se ejecuta:

```
DatabaseConnectionOracle lobjDataBase = new
DatabaseConnectionOracle("DataSource", "user", "password");
```

En el primer parámetro se pasa el origen de datos (por ejemplo en el caso de Oracle 11g Express sería “XE”). En el segundo parámetro se indica el usuario SQL, en el tercer parámetro se pasa la contraseña. Si lo que se desea es una conexión remota a Oracle, se incluiría:

```
DatabaseConnectionOracle lobjDataBase = new
DatabaseConnectionOracle("//192.168.0.17:1521/XE", "user", "password");
```

Siendo 192.168.0.17 la IP del equipo que tiene Oracle Server, siendo 1521 el puerto TCP de comunicación (es el puerto estándar de Oracle), y siendo XE el origen de datos en una instalación Oracle 11g Express.

Para crear la conexión con una base de datos de Firebird se ejecuta:

```
DatabaseConnectionFirebird lobjDataBase = new
DatabaseConnectionFirebird"c:\\Database.fdb", "SYSDBA", "masterkey");
```

En el primer parámetro se pasa la base de datos con su ruta (path) completa. En el segundo parámetro se indica el usuario SQL, en el tercer parámetro se pasa la contraseña. Si lo que se desea es una conexión remota a Firebird, se utilizaría el segundo constructor de la clase:

```
DatabaseConnectionFirebird lobjDataBase = new
DatabaseConnectionFirebird("192.168.0.17", "c:\\Database.fdb", "SYSDBA",
"masterkey", 3050, lobjStruct);
```

Siendo 192.168.0.17 la IP del equipo que tiene Firebird, y siendo 3050 el puerto TCP de comunicación (es el puerto estándar de Firebird).

Para crear la conexión con una base de datos de MySQL se ejecuta:

```
DatabaseConnectionMySQL lobjDataBase = new DatabaseConnectionMySQL ("server
address", "database name", "user", "password");
```

En el primer parámetro se pasa la dirección del servidor MySQL, en el segundo parámetro se pasa el nombre de la base de datos. En el tercer parámetro se indica el usuario SQL, en el cuarto parámetro se pasa la contraseña.

Existe un segundo constructor de la clase al que le podemos añadir el puerto TCP de comunicación:

```
DatabaseConnectionMySQL lobjDataBase = new DatabaseConnectionMySQL ("server
address", "database name", "user", "password", TCP Port);
```

Para crear la conexión con una base de datos de Access se ejecuta:

```
DataConnectionAccess lobjConnection = new
DataConnectionAccess("DataBase.mdb", "Path DataBase", "Password Database");
```

En el primer parámetro se pasa el nombre de archivo Access (.mdb) que contiene la base de datos. En el segundo parámetro se indica la ruta completa de localización del fichero en disco. En el tercer parámetro se indica la Password de Access si la tuviera (se asigna en

Microsoft Access dentro de Seguridad -> Establecer contraseña), si no hay contraseña se pasa String.Empty.

Una vez creada la conexión, e independientemente de que sea de tipo SQL Server, Oracle, Firebird, MySQL, o Access, existen dos funciones para ejecutar comandos, el primero llamado SQLSelectExecute y exclusivo para obtener datos de la base de datos (mediante un comando SQL estándar), como por ejemplo:

```
lstrSQLCommand = "SELECT Name, Lastname FROM Staff";
DataTable lobjData = lobjConnection.SQLSelectExecute(lstrSQLCommand);

foreach (DataRow dr in lobjData.Rows)
{
   Console.WriteLine("Staff Name: " + dr["Name"].ToString() + " -- Last
name: " + dr["Lastname"].ToString());
}
```

Un segundo comando llamado SQLCommandExecute para ejecutar el resto de órdenes SQL estándar (INSERT, UPDATE, DELETE, etcétera). Por ejemplo para insertar datos en la base de datos podemos utilizar:

```
string lstrSQLCommand = "INSERT INTO Staff (IDPerson, Name, Lastname,
Street, City) ";
lstrSQLCommand += " VALUES (30, 'John', 'Smith', '5th', 'New York')";
Boolean lblnInsertOne = lobjConnection.SQLCommandExecute(lstrSQLCommand);

lstrSQLCommand = "INSERT INTO Staff (IDPerson, Name, Lastname, Street,
City) ";
lstrSQLCommand += " VALUES (40, 'David', 'Johnson', '8th', 'Boston')";
Boolean lblnInsertTwo = lobjConnection.SQLCommandExecute(lstrSQLCommand);
if (lblnInsertOne)
{
   Console.WriteLine("The first row has been inserted sucessfully");
 }
 if (lblnInsertTwo)
 {
    Console.WriteLine("The second row has been inserted sucessfully");

 }
```

Si deseamos borrar datos de la base de datos podemos utilizar:

```
lstrSQLCommand = "DELETE FROM STAFF WHERE IDPERSON=30";
Boolean lblnDeleteOne = lobjConnection.SQLCommandExecute(lstrSQLCommand);

lstrSQLCommand = "DELETE FROM STAFF WHERE IDPERSON=40";
Boolean lblnDeleteTwo = lobjConnection.SQLCommandExecute(lstrSQLCommand);
if (lblnDeleteOne)
{
    Console.WriteLine("The first row has been deleted sucessfully");
}
if (lblnDeleteTwo)
{
    Console.WriteLine("The second row has been deleted sucessfully");
```

```
}
```

De la misma forma podemos ejecutar otros comandos SQL estándar existente, como por ejemplo UPDATE.

Una vez creada la conexión, se ejecutan en serie todos los comandos SQL necesarios, y al finalizar el último SIEMPRE se debe de cerrar la conexión, ya que si no se cierra permanecerán abiertas en el servidor de datos y terminará por bloquearse. Para cerrar la conexión creada se utiliza:

```
lobjConnection.CloseConnection();
```

En los ejemplos que acompañan a la librería (carpeta “Samples” que viene dentro del archivo ZIP SwanCSharp.zip descargado) existe un ejemplo de un proyecto de Acceso a Datos que opera contra Access (la base de datos está incluida) y contra SQL Server (se requiere tener SQL Server instalado y la base de datos y tabla creadas).

Dentro de las clases de conexión existen varias funciones predefinidas que nos puede ayudar a realizar pequeños procesos rutinarios en bases de datos sin necesidad de utilizar de desarrollar código fuente. A continuación pasamos a describir dichas funciones.

10.4.1.1 Transacciones

En todos los constructores de la clase DataAccess (SQL Server, Oracle, Firebird, MySQL, y Access) existen los métodos BeginTransaction, CommitTransaction y RollBackTransaction que permiten gestionar transacciones de datos en la clase “DataAccess”.

Una vez creada una conexión, para crear una transacción, se ejecuta el método:

```
lobjConnection.BeginTransaction();
```

Una vez ejecutados la serie de comandos SQL deseados, para confirmar la transacción y que sea efectiva se ejecuta el método:

```
lobjConnection.CommitTransaction();
```

Si lo que se desea es anular la transacción y que no sea ejecutada se utiliza el método:

```
lobjConnection.RollBackTransaction();
```

10.4.1.2 CalculateNextIntegerValue

Esta función nos permite calcular el próximo valor ID a aplicar a un registro nuevo que se vaya a insertar. Como primer parámetro se pasa el nombre de la tabla en la que vamos a insertar el registro; como segundo parámetro se pasa el nombre del campo que tiene la característica de ser el campo ID. La función accede a esa tabla y nos devuelve un valor “Int32” con el próximo valor ID que llevará el próximo registro a insertar.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.DataAccess;
```

Un ejemplo de llamada a la función es el siguiente:

```
DataConnectionSQLServer lobjConnectionLogin = new
DataConnectionSQLServer("PC-NAME\\SQL INSTANCE", "Database",
DatabaseManager.SQLServer);

Int32 lintNumber = lobjConnectionLogin.CalculateNextIntegerValue("Staff",
"IDPerson");
```

10.4.1.3 CheckExistingDataInteger

Esta función comprueba si un dato entero elegido existe en un campo entero determinado dentro de una tabla determinada de la base de datos.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.DataAccess;
```

Un ejemplo de llamada a la función es el siguiente:

```
DataConnectionSQLServer lobjConnectionLogin = new
DataConnectionSQLServer("PC-NAME\\SQL INSTANCE", "Database",
DatabaseManager.SQLServer);

Boolean lblnResult = lobjConnectionLogin.CheckExistingDataInteger("Table
Name", "Field Name", "Data Integer to Compare");
```

10.4.1.4 CheckExistingDataString

Esta función comprueba si un dato string elegido existe en un campo string determinado dentro de una tabla determinada de la base de datos.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.DataAccess;
```

Un ejemplo de llamada a la función es el siguiente:

```
DataConnectionSQLServer lobjConnectionLogin = new
DataConnectionSQLServer("PC-NAME\\SQL INSTANCE", "Database",
DatabaseManager.SQLServer);

Boolean lblnResult = lobjConnectionLogin.CheckExistingDataString("Table
Name", "Field Name", "Data String to Compare");
```

10.4.2 DatabaseStructureSQLServer, DatabaseStructureOracle, DataBaseStructureFirebird, DataBaseStructureMySQL y DatabaseStructureAccess

La finalidad de la clase es crear dinámicamente (en tiempo de ejecución) bases de datos de SQL Server, Oracle, Firebird, MySQL o Access. Esta clase es muy útil para desarrollar aplicaciones que implícitamente pueden crear y/o actualizar la estructura de sus propias bases de datos, para facilitar la actualización de versiones antiguas según se desarrollan versiones nuevas.

También se puede utilizar para crear aplicaciones propietarias que puedan crear una base de datos o actualizarla.

En el caso de la clase “DatabaseStructureOracle” NO es necesario crear previamente la base de datos donde se añadirán las tablas de datos, ya que Oracle no opera con bases de datos sino con esquemas de usuarios. Para este tipo de bases de datos se ignora el dato “Identity” en “DBStruct” ya que no existe el tipo de campo “Autonumérico”.

En el caso de la clase “DatabaseStructureFirebird” y “DatabaseStructureMySQL” es necesario crear previamente la base de datos donde se añadirán las tablas de datos, ya que no la van a crear. Para este tipo de bases de datos se ignora el dato “Identity” en “DBStruct” ya que no existe el tipo de campo “Autonumérico”.

En el caso de la clase “DatabaseStructureAccess” es necesario crear previamente el archivo MDB vacío, ya que “DatabaseStructureAccess” no lo va a crear, simplemente utiliza uno sin estructura y le crea dentro la estructura deseada.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.DataAccess;
```

Para crear una estructura de datos en SQL Server se ejecuta:

```
DatabaseStructureSQLServer lobjDataBase  = new
DatabaseStructureSQLServer("PC-NAME\\INSTANCE", "DataBase", lobjStruct);
```

Para crear una estructura de datos en Oracle se ejecuta:

```
DatabaseStructureOracle lobjDataBase = new DatabaseStructureOracle("XE",
"system", "paginaweb", lobjStruct);
```

Para crear una estructura de datos en un archivo Firebird se ejecuta:

```
DatabaseStructureFirebird lobjDataBase = new
DatabaseStructureFirebird("c:\\Database.fdb", "SYSDBA", "masterkey",
lobjStruct);
```

ó

```
DatabaseStructureFirebird lobjDataBase = new
DatabaseStructureFirebird("localhost", "c:\\Database.fdb", "SYSDBA",
"masterkey", 3050, lobjStruct);
```

Para crear una estructura de datos en un archivo MySQL se ejecuta:

```
DatabaseStructureMySQL lobjDataBase = new DatabaseStructureMySQL("server
address", "Database name", "user", "password", lobjStruct);
```

ó

```
DatabaseStructureMySQL lobjDataBase = new DatabaseStructureMySQL("server
address", "Database name", "user", "password", TCP Port, lobjStruct);
```

Para crear una estructura de datos en un archivo Access se ejecuta:

```
DatabaseStructureAccess lobjDataBase = new
DatabaseStructureAccess("DataBase.mdb", "", "", lobjStruct);
```

En ambos casos el último parámetro, en el que estamos pasando "lobjStruct", se refiere a la estructura DBStruct existente en la clase DataAccess. Dicha estructura se compone de las variables TableName, FieldName, FieldType, PrimaryKey, Identity, y NotNull. Creando un array de DBStruct, definimos la estructura deseada para nuestra base de datos. Posteriormente se pasa el array DBStruct al constructor de las clases “DatabaseStructure” y dicha clase se encargará de crear toda la estructura.

A continuación se muestra un ejemplo de creación de una estructura:

```
DBStruct[] lobjStruct = new DBStruct[8];

lobjStruct[0].TableName = "Staff";
lobjStruct[0].FieldName = "IDPerson";
lobjStruct[0].FieldType = "int";
lobjStruct[0].PrimaryKey = true;
lobjStruct[0].Identity = true;
lobjStruct[0].NotNull = true;

lobjStruct[1].TableName = "Staff";
lobjStruct[1].FieldName = "Name";
lobjStruct[1].FieldType = "varchar(20)";
lobjStruct[1].PrimaryKey = true;
lobjStruct[1].Identity = false;
lobjStruct[1].NotNull = true;

lobjStruct[2].TableName = "Staff";
lobjStruct[2].FieldName = "Lastname";
lobjStruct[2].FieldType = "varchar(20)";
lobjStruct[2].PrimaryKey = false;
lobjStruct[2].Identity = false;
lobjStruct[2].NotNull = true;
```

```
lobjStruct[3].TableName = "Staff";
lobjStruct[3].FieldName = "Street";
lobjStruct[3].FieldType = "varchar(50)";
lobjStruct[3].PrimaryKey = false;
lobjStruct[3].Identity = false;
lobjStruct[3].NotNull = true;

lobjStruct[4].TableName = "Staff";
lobjStruct[4].FieldName = "City";
lobjStruct[4].FieldType = "varchar(50)";
lobjStruct[4].PrimaryKey = false;
lobjStruct[4].Identity = false;
lobjStruct[4].NotNull = true;

lobjStruct[5].TableName = "Auxiliary";
lobjStruct[5].FieldName = "IDPerson";
lobjStruct[5].FieldType = "int";
lobjStruct[5].PrimaryKey = true;
lobjStruct[5].Identity = false;
lobjStruct[5].NotNull = true;

lobjStruct[6].TableName = "Auxiliary";
lobjStruct[6].FieldName = "DischargeDate";
lobjStruct[6].FieldType = "datetime";
lobjStruct[6].PrimaryKey = false;
lobjStruct[6].Identity = false;
lobjStruct[6].NotNull = false;

lobjStruct[7].TableName = "Auxiliary";
lobjStruct[7].FieldName = "Salary";
lobjStruct[7].FieldType = "int";
lobjStruct[7].PrimaryKey = false;
lobjStruct[7].Identity = false;
lobjStruct[7].NotNull = false;

// SQLServer
DatabaseStructureSQLServer lobjDataBase  = new
DatabaseStructureSQLServer("PC-NAME\\INSTANCE", "DataBase", lobjStruct);

// Access
DatabaseStructureAccess lobjDataBase = new
DatabaseStructureAccess("DataBase.mdb", "", "", lobjStruct);
```

En el ejemplo anterior se está definiendo una estructura de dos tablas (Staff, Auxiliary) para la base de datos DataBase. Para la primera tabla (Staff) se definen dos campos clave principal y otros tres campos normales. Para la segunda tabla (Auxiliary) se define un campo clave principal y otros dos campos estándar. Con este ejemplo estamos creando dinámicamente desde código fuente la estructura de una base de datos tanto en Access como en SQLServer.

A la hora de crear y definir la estructura DBStruct solamente hay que tener en cuenta algunas normas:

- Se pueden crear simultáneamente todas las tablas diseñadas en un único DBStruct y en una única ejecución de la case DatabaseStructure, pero los campos de cada tabla

deben ir correlativos dentro de los elementos del DBStruct, es decir, no se pueden intercalar campos de diferentes tablas.

- Cada tabla debe de tener como mínimo un campo clave principal, si no la tabla no se creará, pudiendo tener más de un campo clave principal.
- El campo "FieldType" se define utilizando los mismos tipos de datos que existen en SQL Server y en Access.
- Las normas de creación de las tablas son las mismas que en SQL Server y Access, por ejemplo, no se puede crear un campo clave principal si no está a "true" la variable "NotNull" para dicho campo, o no se puede asignar la propiedad Identity a un campo que no sea "int".

En los ejemplos que acompañan a la librería (carpeta “Samples” que viene dentro del archivo ZIP SwanCSharp.zip descargado) existe un ejemplo de un proyecto "DatabaseCreation" que opera contra Access (la base de datos está incluida) y contra SQL Server (se requiere tener SQL Server instalado y la base de datos y tabla creadas).

10.4.3 DataExport

La clase "DataExport" permite exportar un objeto "DataTable" a Excel en formato XLS o CSV, permitiendo exportar datos de Access o SQL Server obtenidos mediante la clase DataConnection o cualquier otro DataTable cargado manualmente u obtenido de otra forma.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.DataAccess;
```

Un código fuente de ejemplo puede ser:

```
// Open Connection
DataConnectionSQLServer lobjConnection = new DataConnectionSQLServer("PC-
NAME\\INSTANCE_SQL", "DataBase", DatabaseManager.SQLServer);

//SELECT
string lstrSQLCommand = "SELECT * FROM Staff";
DataTable lobjData = lobjConnection.SQLSelectExecute(lstrSQLCommand);

// Xls
DataExport.ExportDatabaseToExcel(lobjData, ExcelFormat.Excel,
"Test_Xls.xls");
// Csv
DataExport.ExportDatabaseToExcel(lobjData, ExcelFormat.CSV,
"Test_CSV.xls");
```

En los ejemplos que acompañan a la librería (carpeta “Samples” que viene dentro del archivo ZIP SwanCSharp.zip descargado) existe un ejemplo de un proyecto "DataExport" que permite exportar unos datos a Excel (CSV o XLS).

10.4.4 Utilities

El nombre de espacio "DataAccess" tiene una clase “Utilities” que engloba a todas aquellas funciones que no requieren un constructor para ser llamadas. Dichas funciones nos permiten disponer de ciertas utilidades en el apartado de las bases de datos.

10.4.4.1 DataGridViewToDataTable

La función "DataGridViewToDataTable" nos permite convertir un control “DataGridView” de Windows.Forms en un datatable.

En la cabecera del procedimiento se añade la referencia a la clase:

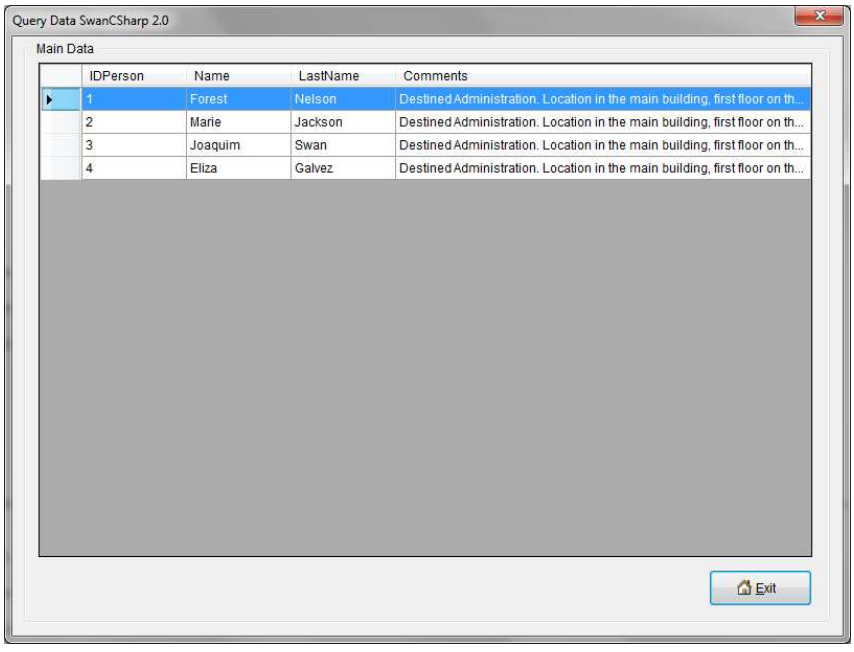
```
using SwanCSharp.DataAccess;
```

Un código fuente de ejemplo puede ser:

```
DataTable ldatData = Utilities.DataGridViewToDataTable(pobjDataGridView);
```

10.4.4.2 ShowQueryDataModifyWindow

La función "ShowQueryDataModifyWindow" nos permite mostrar una ventana con un "Grid" de datos, modificar las celdas deseadas, y nos devuelve un "DataTable" con los datos actualizados:



La función "ShowQueryDataModifyWindow" espera recibir siete parámetros que son los siguientes: el DataTable con los datos; un array de enteros con las columnas del datatable que no se desean mostrar (si se quieren visualizar todas, se pasa "null", y el número de cada columna es so ordinal, siendo "0" la primera columna); el titulo deseado para el formulario que se va a mostrar; el título deseado para el "frame" que se muestra en el

formulario; el tamaño de la ventana (mediante el enumerado QueryWindowSize, eligiendo entre Small, Medium, High); el idioma del formulario (mediante el enumerado InterfaceLanguage, eligiendo entre español e inglés); y por último un valor "true" o "false" si se desea que la última columna se expanda automáticamente al ancho total del grid, o no.

Después de ejecutar la función aparecerá un formulario Windows mostrando los datos en pantalla. El usuario puede modificar los datos de las celdas deseadas, al pulsar en "Exit" la función devolverá un DataTable con todos los datos modificados.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.DataAccess;
```

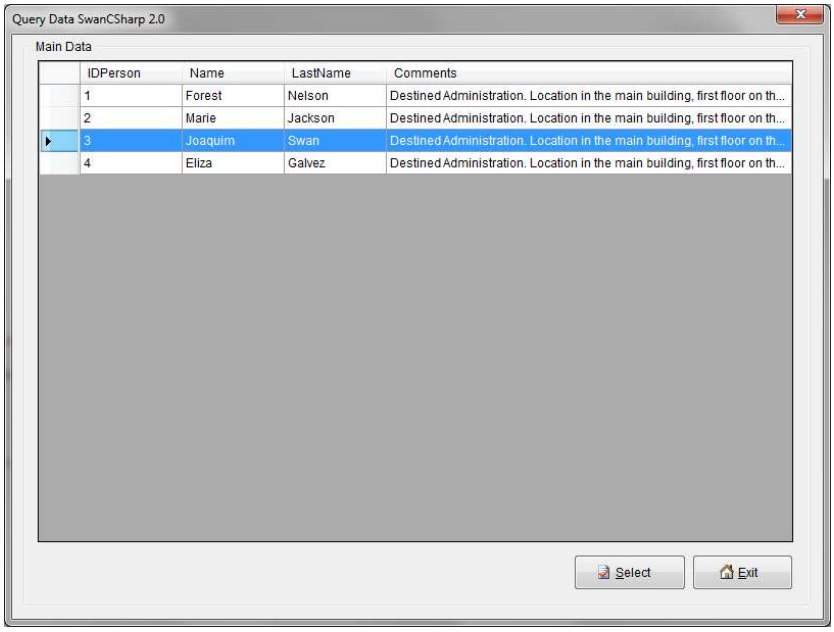
Un código fuente de ejemplo puede ser:

```
DataConnectionAccess lobjConnection = new
DataConnectionAccess("DataBase.mdb", "", "", DatabaseManager.Access);
string lstrSQL = "SELECT * FROM Staff";
DataTable ldatData = lobjConnection.SQLSelectExecute(lstrSQL);
lobjConnection.CloseConnection();

int[] lintHiddenColumns = new int[2];

lintHiddenColumns[0] = 0;
lintHiddenColumns[1] = 3;

ldatData = Utilities.ShowQueryDataModifyWindow(ldatData, lintHiddenColumns,
"Query Data SwanCSharp", " Main Data ", QueryWindowSize.Medium,
InterfaceLanguage.English, true);
```

En los ejemplos que acompañan a la librería (carpeta “Samples” que viene dentro del archivo ZIP SwanCSharp.zip descargado) existe un ejemplo de un proyecto "DataQueryAccess" que opera contra Access (la base de datos está incluida).

10.4.4.3 ShowQueryDataWindow

La función "ShowQueryDataWindow" nos permite mostrar una ventana con un "Grid" de datos y nos devuelve un "DataRow" con la fila que se ha seleccionado en la ventana. Una captura de pantalla es la siguiente:



La función "ShowQueryDataWindow" espera recibir siete parámetros que son los siguientes: el DataTable con los datos; un array de enteros con las columnas del datatable que no se desean mostrar (si se quieren visualizar todas, se pasa "null", y el número de cada columna es so ordinal, siendo "0" la primera columna); el titulo deseado para el formulario que se va a mostrar; el título deseado para el "frame" que se muestra en el formulario; el tamaño de la ventana (mediante el enumerado QueryWindowSize, eligiendo entre Small, Medium, High); el idioma del formulario (mediante el enumerado InterfaceLanguage, eligiendo entre español e inglés); y por último un valor "true" o "false" si se desea que la última columna se expanda automáticamente al ancho total del grid, o no.

Después de ejecutar la función aparecerá un formulario Windows mostrando los datos en pantalla. El usuario puede seleccionar una línea mediante doble-clic sobre la fila, o un clic sobre la fila y pulsando en el botón "Seleccionar". La función devolverá un "DataRow" con los datos de la fila elegida.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.DataAccess;
```

Un código fuente de ejemplo puede ser:

```
DataConnectionAccess lobjConnection = new
DataConnectionAccess("DataBase.mdb", "", "", DatabaseManager.Access);
string lstrSQL = "SELECT * FROM Staff";
DataTable ldatData = lobjConnection.SQLSelectExecute(lstrSQL);
lobjConnection.CloseConnection();

DataRow ldarRow;
int[] lintHiddenColumns = new int[2];

lintHiddenColumns[0] = 0;
lintHiddenColumns[1] = 4;
```

```
ldarRow = Utilities.ShowQueryDataWindow(ldatData, lintHiddenColumns, "Query
Data SwanCSharp 2.0", " Main Data ", QueryWindowSize.Medium,
InterfaceLanguage.English, true);
```

En los ejemplos que acompañan a la librería (carpeta “Samples” que viene dentro del archivo ZIP SwanCSharp.zip descargado) existe un ejemplo de un proyecto "DataQueryAccess" que opera contra Access (la base de datos está incluida).

Nota importante: En el caso de los formularios personalizados “SwanCSharp” creados desde el espacio de nombres “SwanCSharp_Controls”, será necesario utilizar la clase “WindowDataQuery” existente en ese espacio de nombres en lugar de esta función “ShowQueryDataWindow”.

10.5 SwanCSharp.Directories

La clase "Directories" incorpora una serie de funciones de asistencia con el manejo de directorios en rutas de disco.

10.5.1 CopyDirectory

La función CopyDirectory permite copiar un directorio origen en un directorio destino diferente. La función recibe dos parámetros <string>, el primero con la ruta origen, el segundo con la ruta destino, y un tercer parámetro booleano donde se informa si se desea que el copiado sea recursivo o no (recursivo además de copiar la carpeta y sus archivos, copia todas las carpetas junto con sus archivos que dependan del directorio original). La función devuelve un valor booleano informado si el proceso ha sido ejecutado o no.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
Boolean lblnResult  = Directories.CopyDirectory(“c:\testSource”,
“c:\testTarget”, true);
```

10.5.2 GetFolderNamesRecursively

La función “GetFolderNamesRecursively”, dada una ruta con una carpeta de inicio, nos devuelve un array de string con todas las carpetas existentes en su estructura en árbol recursiva.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
string[] lstrFolders  =
Directories.GetFolderNamesRecursively(“c:\\testSource”);
```

10.6 SwanCSharp.Encryption

La clase Encryption incorpora una clase para cifrar datos y varias funciones que permiten mediante una llamada sencilla cifrar y descifrar información en bloques de información. El sistema de cifrado utilizado es AES, pudiendo cifrar mediante HASH MD5 o SHA1, y utilizando dos palabras clave.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Encryption;
```

Una línea de código de ejemplo puede ser:

```
AES lobjAES = new AES("ourpassphrase", "oursaltvalue", HashFunction.SHA1,
2, "2hs@3TY9F4fz5%tv", KeySize.Key256);
```

El primer parámetro del constructor se refiere a la palabra clave que se utilizará para cifrar que es un "string" con valores ASCII elegidos libremente por el desarrollador, el segundo parámetro se corresponde a la palabra que se utilizara en combinación con el primer parámetro para generar el cifrado (también es un "string" libre formado por valores ASCII). El tercer parámetro se alimenta del enumerado HashFunction existente en la clase y nos permite seleccionar si el cifrado se desea realizar mediante Hash MD5 o SHA-1 (el segundo ofrece mayor seguridad). El cuarto parámetro se utiliza para informar el número de iteraciones deseadas para generar el cifrado, con 2 iteraciones es suficiente. El quinto parámetro hace referencia al vector de inicialización que siempre tiene que ser un "string" ASCII de 16 caracteres de longitud exactos. El sexto parámetro hace referencia al enumerado KeySize de la clase y permite seleccionar la longitud de la clave generada, siendo posible elegir entre 128, 192, o 256 (cuanto más grande sea la clave, más seguro es el cifrado).

Cuando se cifran datos con la clase SwanCSharp.Encryption, los seis parámetros utilizados para el cifrado, serán los seis parámetros que habrá que utilizar para descifrar esa misma cadena.

10.6.1 BytesEncryptToFile

Se utiliza para cifrar un array de bytes y el resultado se graba directamente en un fichero determinado en una ruta determinada.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Encryption;
```

Se construye la clase de cifrado:

```
AES lobjAES = new AES("ourpassphrase", "oursaltvalue", HashFunction.SHA1,
2, "2hs@3TY9F4fz5%tv", KeySize.Key256);
```

Una línea de código de ejemplo puede ser:

```
Boolean lblnResult = lobjAES.BytesEncryptToFile(pobjBytes,
"FileDecrypted.txt", "c:\\test", true);
```

10.6.2 FileDecrypt

Se utiliza para descifrar un archivo que previamente fue cifrado con la función FileEncrypt. Espera recibir cinco parámetros donde los primeros cuatro parámetros recogen la información del fichero origen y del fichero destino y sus rutas de acceso, y el quinto parámetro recibe un valor "true" si se desea que el fichero destino sobrescriba al que pueda existir previamente, y false si no se desea que sobrescriba al fichero.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Encryption;
```

Se construye la clase de cifrado (el usuario debe informar los mismos seis parámetros utilizados para cifrar el fichero que se desea descifrar):

```
AES lobjAES = new AES("ourpassphrase", "oursaltvalue", HashFunction.SHA1,
2, "2hs@3TY9F4fz5%tv", KeySize.Key256);
```

Una línea de código de ejemplo puede ser:

```
Boolean lblnResult = lobjAES.FileDecrypt("FileEncrypted.txt", "",
"FileDecrypted.txt", "", true);
```

10.6.3 FileDecryptToBytes

Se utiliza para descifrar en un array de bytes un archivo que previamente fue cifrado con la función FileEncrypt o BytesEncryptToFile. Espera recibir dos parámetros con el nombre del fichero origen y la ruta del fichero origen.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Encryption;
```

Se construye la clase de cifrado (el usuario debe informar los mismos seis parámetros utilizados para cifrar el fichero que se desea descifrar):

```
AES lobjAES = new AES("ourpassphrase", "oursaltvalue", HashFunction.SHA1,
2, "2hs@3TY9F4fz5%tv", KeySize.Key256);
```

Una línea de código de ejemplo puede ser:

```
byte[] lobjBytes = lobjAES.FileDecryptToBytes("FileEncrypted.txt",
"c:\\test");
```

10.6.4 FileEncrypt

Se utiliza para cifrar un archivo. Espera recibir cinco parámetros donde los primeros cuatro parámetros recogen la información del fichero origen y del fichero destino y sus rutas de acceso, y el quinto parámetro recibe un valor "true" si se desea que el fichero destino sobrescriba al que pueda existir previamente, y false si no se desea que sobrescriba al fichero.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Encryption;
```

A continuación se construye la clase de cifrado (el usuario debe elegir los seis parámetros que desea utilizar para cifrar el fichero, esos mismos parámetros deberán ser utilizados para el descifrado):

```
AES lobjAES = new AES("ourpassphrase", "oursaltvalue", HashFunction.SHA1,
2, "2hs@3TY9F4fz5%tv", KeySize.Key256);
```

Una línea de código de ejemplo puede ser:

```
Boolean lblnResult = lobjAES.FileEncrypt("FileToEncrypt.txt", "",
"FileEncrypted.txt", "", true);
```

10.6.5 StringDecrypt

Se utiliza para descifrar un "string" que previamente fue cifrado con la función StringEncrypt. Se pasa como parámetro el "string" a descifrar, y la función devuelve en el segundo parámetro una referencia con el "string" descifrado.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Encryption;
```

Se construye la clase de cifrado (el usuario debe informar los mismos seis parámetros utilizados para cifrar el "string" que se desea descifrar):

```
AES lobjAES = new AES("ourpassphrase", "oursaltvalue", HashFunction.SHA1,
2, "2hs@3TY9F4fz5%tv", KeySize.Key256);
```

Una línea de código de ejemplo puede ser:

```
string lstrDecryptedText ="";
Boolean lblnResult = lobjAES.StringDecrypt(lstrEncrypted, out
lstrDecryptedText);
```

10.6.6 StringEncrypt

Se utiliza para cifrar un "string". Se pasa como parámetro el "string" a cifrar, y la función devuelve el "string" cifrado.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Encryption;
```

A continuación se construye la clase de cifrado (el usuario debe elegir los seis parámetros que desea utilizar para cifrar el "string", esos mismos parámetros deberán ser utilizados para el descifrado):

```
AES lobjAES = new AES("ourpassphrase", "oursaltvalue", HashFunction.SHA1,
2, "2hs@3TY9F4fz5%tv", KeySize.Key256);
```

Una línea de código de ejemplo puede ser:

```
string lstrEncrypted = lobjAES.StringEncrypt("Test of Encryption");
Console.WriteLine("Encrypted Text: " + lstrEncrypted);
```

10.7 SwanCSharp.Files

La clase Files incorpora una serie de funciones de asistencia con el manejo de archivos y rutas de disco.

10.7.1 CheckPathEndsBackslash

La función recibe un parámetro <string> con una ruta de un disco, comprueba si la ruta acaba en contrabarra (\ o backslash), si no la tiene se la agrega y devuelve el nuevo <string>, si ya tiene la barra devuelve el mismo <string>.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
lstrFilesPath = Files.CheckPathEndsBackslash(lstrFilesPath);
```

10.7.2 CheckPathEndsSlash

La función recibe un parámetro <string> con una ruta de un disco, comprueba si la ruta acaba en barra (/ o slash), si no la tiene se la agrega y devuelve el nuevo <string>, si ya tiene la barra devuelve el mismo string.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
lstrFilesPath = Files.CheckPathEndsSlash(lstrFilesPath);
```

10.7.3 FileMove

La finalidad de la función es mover un archivo origen a un lugar destino (indicando el nombre del archivo movido). Espera recibir tres parámetros y devuelve un valor bool que informa si se ha movido o no. En el primer parámetro se indica el nombre de archivo a mover con su ruta completa; en el segundo parámetro se indica el nombre de archivo destino con su ruta completa; en el tercer parámetro se informa mediante un valor bool si se desea sobrescribir el archivo destino en caso de que existiera.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
Boolean lblnFileMoved = Files.FileMove("c:\\test\\test.txt",
"c:\\NewTest\\test.txt", true);
```

10.7.4 FileToArrayBytes

La finalidad de la función es leer un fichero y convertir todo su contenido en un array de bytes. Espera recibir dos parámetros, el primero con la ruta donde se encuentra el fichero, y

el segundo con el nombre del fichero. La función devuelve un objeto array de bytes con el contenido del fichero.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
byte[] lobjFileNameByte = Files.FileToArrayBytes(pstrSourcePath,
pstrFileName);
```

10.7.5 GZIPUncompressFile

La finalidad de la función es descomprimir un fichero Gzip dado en otro fichero destino pasado por parámetro.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
Boolean lblnResult = Files.GZIPUncompressFile(pstrFileGZ,
pstrTargetFileName);
```

10.7.6 RemoveInvalidCharsFileName

La finalidad de la función es eliminar de un string todos aquellos caracteres que no pueden formar parte del nombre de un fichero. En el primer parámetro recibe el nombre del fichero, en el segundo parámetro se debe indicar por qué carácter se deben sustituir aquellos que sean considerados inválidos. La función devuelve un string con la cadena convertida.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
string lstrFileNameNew = Files.RemoveInvalidCharsFileName(pstrFileName,
'_');
```

10.7.7 SaveArrayBytesToFile

La finalidad de la función es volcar el contenido de un array de bytes en un fichero concreto grabando en la ubicación deseada. La función recibe cuatro parámetros, el primero espera recibir el array de bytes, el segundo la longitud del array que será el tamaño del archivo cuando sea generado, en el tercero se recibe la ruta donde se va a almacenar el archivo, y en el cuarto parámetro se recibe el nombre que va a tener el archivo. La función genera el archivo en la ruta deseada, y con la longitud pasada, y devuelve verdadero si se ha ejecutado el proceso con éxito, y falso si no se ha podido generar el archivo.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
Boolean lblnSaved = Files.SaveArrayBytesToFile(lobjDataBytes,
receivedBytesLen, pstrTargetPath, pstrFileName);
```

10.7.8 SeparateFileNameAndPath

La finalidad de la función es recibir una ruta completa de un fichero (ruta + nombre de fichero) y separar en dos variables diferentes la ruta y el nombre del fichero. En el primer parámetro se informa con el nombre de fichero + ruta, y en el segundo y tercer parámetro se recibe por referencia una variable con la ruta y otra variable con el nombre de fichero.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
string lstrPathFileName = "";
string lstrFileName = "";
Files.SeparateFileNameAndPath(@"c:\test\filetest.txt", ref
lstrPathFileName, ref lstrFileName);
```

10.7.9 UnZip

La finalidad de la función es descomprimir un fichero ZIP. En el primer parámetro se incluye el fichero zip a descomprimir con su ruta completa (obligatorio incluir la ruta), y en el segundo parámetro se incluye la ruta destino donde se grabarán los ficheros descomprimidos.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
Boolean lblnResult = UnZip(@"C:\Zip\Prueba.zip", @"C:\Unzip");
```

En los ejemplos que acompañan a la librería (carpeta “Samples” que viene dentro del archivo ZIP SwanCSharp.zip descargado) existe un ejemplo de un proyecto de descompresión de ficheros.

10.7.10 Zip

La finalidad de la función es comprimir en un único fichero ZIP uno o más archivos localizados en una ruta o en varias rutas diferentes. En el primer parámetro se incluye el fichero zip con su ruta completa (obligatorio incluir la ruta), y en el segundo parámetro se incluye un array de strings donde se define cada archivo a comprimir. Para poder generar el fichero ZIP es necesario que cada fichero incluido en el array exista, e incluya la ruta completa de acceso al fichero.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
string[] lstrFileNames = new string[2];
lstrFileNames[0] = "C:\\Test\\File1.txt";
lstrFileNames[1] = "C:\\Test\\File2.txt";

Boolean lblnResult = Files.Zip(@"C:\Zip\Prueba.zip", lstrFileNames);
```

En los ejemplos que acompañan a la librería (carpeta “Samples” que viene dentro del archivo ZIP SwanCSharp.zip descargado) existe un ejemplo de un proyecto de compresión de ficheros.

10.8 SwanCSharp.Imaging

La clase Imaging incorpora una serie de funciones de asistencia con el manejo de ficheros de imágenes.

10.8.1 BitmapResize

La finalidad de la función es recoger una imagen origen, y a partir de ella crear otra imagen con las dimensiones pasadas por parámetro. En el primer parámetro se espera recibir la imagen origen, en el segundo parámetro el ancho en píxeles (integer), y en el tercer

parámetro el alto en píxeles (integer). La función devuelve un objeto “Bitmap” con una nueva imagen con las dimensiones especificadas.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
Bitmap lobjNewImage = Imaging.BitmapResize(lobjImage, 640, 480);
```

10.8.2 BitmapToMemoryStream

La finalidad de la función es recoger una imagen origen y convertirla al tipo de datos MemoryStream. En el primer parámetro se espera recibir la imagen origen, en el segundo parámetro se espera recibir el formato de la imagen volcada al stream. La función devuelve el objeto MemoryStream cargado.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
MemoryStream lobjStream = Imaging.BitmapToMemoryStream(lobjImage,
System.Drawing.Imaging.ImageFormat.Jpeg);
```

10.8.3 BitmapToUnsafeBytes

La finalidad de la función es recoger una imagen origen y convertirla al tipo de datos Unsafe bytes (byte[]). En el primer parámetro se espera recibir la imagen origen. La función devuelve el objeto “byte unsafe” cargado. Esta función puede ser útil cuando necesitamos pasar una imagen a una función creada en C++.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
byte[] lobjUnsafeBytes = Imaging.BitmapToUnsafeBytes(lobjImage);
```

10.8.4 ByteArrayToBitmap

La finalidad de la función es recoger un array de bytes y convertirlo al tipo de datos Bitmap; en el primer parámetro se recoge el array de bytes, y la función devuelve un objeto “Bitmap”.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
Bitmap lobjBitmap = Imaging.ByteArrayToBitmap(parrByteArray);
```

10.8.5 ByteArrayToIcon

La finalidad de la función es recoger un array de bytes y convertirlo al tipo de datos Icon; en el primer parámetro se recoge el array de bytes, y la función devuelve un objeto “Icon”.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
Icon lobjIcon = Imaging.ByteArrayToIcon(parrByteArray);
```

10.8.6 ByteArrayToImage

La finalidad de la función es recoger un array de bytes y convertirlo al tipo de datos Image; en el primer parámetro se recoge el array de bytes, y la función devuelve un objeto “Image”.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
Image lobjImage = Imaging.ByteArrayToImage(parrByteArray);
```

10.8.7 CompareImages

La finalidad de la función es comparar dos imágenes pasadas por parámetro devolviendo true si son idénticas y false si no lo son.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
bool lblnEqual = Imaging.CompareImages(pobjBitmap1, pobjBitmap2);
```

10.8.8 ConvertBase64ToByteArray

La finalidad de la función es recoger un “String” con formato Base64 y volcarlo sobre un array de bytes.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
byte[] lobjBytes = Imaging.ConvertBase64ToByteArray(pstrFile64);
```

10.8.9 ConvertBase64ToFile

La finalidad de la función es recoger un “String” con formato Base64 y volcarlo sobre un fichero dado.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
Imaging.ConvertBase64ToFile(pstrFile64, pstrTargetFile,
pstrPathTargetFile);
```

10.8.10 ConvertFileToBase64

La finalidad de la función es recoger un fichero y convertirlo al tipo de datos “String” con formato Base64.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
string lstrImage64 = Imaging.ConvertFileToBase64(pstrFilename,
pstrPathFile);
```

10.8.11 ConvertBytesToBase64

La finalidad de la función es recoger un array de bytes y convertirlo al tipo de datos “String” con formato Base64.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
string lstrImage64 = Imaging.ConvertBytesToBase64(pobjBytesArray);
```

10.8.12 ConvertImageBytesToBase64HTML

La finalidad de la función es recoger un array de bytes de una imagen y convertirlo al tipo de Base64 con formato para incrustar en HTML, permitiéndonos insertar una imagen incrustada en una página web mediante la etiqueta “IMG SRC”.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
string lstrImage64 = Imaging.ConvertImageBytesToBase64HTML(pobjImageBytes);

lobjHTMLFile.WriteLine("<img src=\"" + lstrImage64 + "\"/>");
```

10.8.13 ConvertImageFileToBase64HTML

La finalidad de la función es recoger un fichero de imagen y convertirlo al tipo de Base64 con formato para incrustar en HTML, permitiéndonos insertar una imagen incrustada en una página web mediante la etiqueta “IMG SRC”.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
string lstrImage64 = Imaging.ConvertImageFileToBase64HTML(“Image.jpg”,
“C:\\FilePath”);

lobjHTMLFile.WriteLine("<img src=\"" + lstrImage64 + "\"/>");
```

10.8.14 ConvertTo8BppGrayscale

La finalidad de la función es recoger un fichero de imagen y convertirlo a escala de grises con una profundidad de 8-bit (indexado 256 tonos de gris).

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
Bitmap lobjResultBitmap = Imaging.ConvertTo8BppGrayscale(lobjBitmap);
```

10.8.15 DominantColor

La finalidad de la función es recoger una imagen origen y obtener de ella el color dominante de la totalidad de la imagen. En el primer parámetro recibe la imagen, y en los parámetros dos, tres, y cuatro se pasan por referencia un string por cada canal de color RGB (rojo, verde, azul). La función cargará en los últimos tres parámetros el valor del color dominante desglosado en canales.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
string lstrRed = "";
string lstrGreen = "";
string lstrBlue = "";
Imaging.DominantColor(lobjImage, ref lstrRed, ref lstrGreen, ref lstrBlue);
```

10.8.16 GetDifferenceFromImages

La finalidad de la función es obtener el número de pixeles diferentes que hay en dos imágenes pasadas por parámetro.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
double ldblDifference = Imaging.GetDifferenceFromImages(pobjBitmap1,
pobjBitmap2);
```

10.8.17 IconToByteArray

La finalidad de la función es recoger un objeto “icon” y convertirlo a un array de bytes; en el primer parámetro se recoge el objeto Icon, y la función devuelve un array de bytes.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Una línea de código de ejemplo puede ser:

```
byte[] lobjBytes = Imaging.IconToByteArray(pobjIcon);
```

10.9 SwanCSharp.Internet

La clase “Internet” permite incorporar a los desarrollos funciones de utilidad relacionadas con el mundo de las redes WAN e Internet.

10.9.1 BreakdownFTPString

La finalidad de esta función es la de recoger (mediante string) una cadena de conexión de FTP del estilo a "ftp://user:password@ip_ftp/AccessPath" y extraer de ella cada elemento por separado. La función devuelve un array de string que contiene los siguientes valores:

- Posición 0.- Devuelve el nombre de usuario de acceso al FTP.
- Posición 1.- Devuelve la contraseña de acceso al FTP.
- Posición 2.- Devuelve la IP del servidor FTP.
- Posición 3.- Devuelve la carpeta de acceso al FTP.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Internet;
```

Una línea de código de ejemplo puede ser:

```
string[] lstrFTPData =
Utilities.BreakdownFTPString("ftp://user:password@192.168.1.1/data/files");
```

10.9.2 CheckInternetConnection

La finalidad de esta función es la de comprobar si existe conexión a Internet en el equipo de ejecución.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Internet;
```

Una línea de código de ejemplo puede ser:

```
Boolean lblnConnected = Utilities.CheckInternetConnection();
```

10.9.3 FTPClient

La clase “FTPClient” nos permite incorporar a nuestros desarrollos un potente cliente FTP que nos permitirá subir o bajar archivos de un servidor FTP, así como borrar archivos, ejecutar comandos como LIST, ChangeDirectory, RemoveFile, etcétera.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Internet;
```

Una línea de código de ejemplo puede ser:

```
FTPClient lobjFTPClient = new FTPClient();

lobjFTPClient.port = 21;
lobjFTPClient.Connect(“Server IP”, 21, “User FTP”, “Password FTP”);

string lstrFolder = “files/down/”;

if (lstrFolder != String.Empty)
{
   lobjFTPClient.ChangeDirectory(lstrFolder);
}

//Upload a File
lobjFTPClient.Upload(“File to Upload”, “Remote FileName”);
while (lobjFTPClient.Uploading() > 0)
{
}

//Download a File
lobjFTPClient.Download(“File to Download”, “Local FileName”);
while (lobjFTPClient.Downloading() > 0)
{
}

lobjFTPClient.Disconnect();
```

10.9.4 GetDomainNameFromIP

La finalidad de esta función es la de obtener nombre de dominio asociado a una IP dada.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Internet;
```

Una línea de código de ejemplo puede ser:

```
string lstrDomainName = GetDomainNameFromIP("255.255.255.255");
```

10.9.5 GetFileFromHttp

La finalidad de esta función es la de descargar un fichero subido a Internet por medio de su dirección HTTP completa.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Internet;
```

Una línea de código de ejemplo puede ser:

```
string =
GetFileFromHttp("http://www.domain.com/source_folder/source_file.txt",
"Target_file.txt", @"c:\targetfolder");
```

10.9.6 GetIPFromDomainName

La finalidad de esta función es la de obtener la IP asociada a un nombre de dominio de Internet dado.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Internet;
```

Una línea de código de ejemplo puede ser:

```
string lstrIP = GetIPFromDomainName("http://www.domainname.com");
```

10.9.7 GetWebProxyActive

La finalidad de esta función es la de obtener toda la información del proxy web que pudiera estar activo en el ordenador de ejecución, incluidas credenciales.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Internet;
```

Una línea de código de ejemplo puede ser:

```
WebProxy lobjProxy = GetWebProxyActive("http://www.anydomainname.com");
```

10.9.8 IPToUint32

La finalidad de esta función es la de convertir una dirección IP estándar en string a un tipo de datos Uint32.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Internet;
```

Una línea de código de ejemplo puede ser:

```
Uint32 lUintIP = Utilities.IPToUint32("192.168.1.1");
```

10.9.9 Uint32ToIP

La finalidad de esta función es la de convertir una dirección IP Uint32 en una dirección string estándar.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Internet;
```

Una línea de código de ejemplo puede ser:

```
string lstrIPAddress = Utilities.Uint32ToIP(lUintIPAddress);
```

10.10 SwanCSharp.Logger

La clase “Logger” permite incorporar a los desarrollos una gestión eficaz de un sistema de log, pudiendo publicar en el archivo Log mensajes de error, de aviso, o de información.

10.10.1 Logger

Para poder utilizar la clase Logger se debe de crear el objeto pasando dos parámetros: El primer parámetro es la ruta donde se va a generar cada archivo log (si se deja en blanco, se

utilizará la ruta por defecto de ejecución), y el segundo parámetro el nombre de nuestra aplicación.

Al ejecutar el constructor se creará la carpeta “Log” en la ruta destino elegida, y dentro de esa carpeta serán creados todos los archivos log.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Logger;
```

Para crear el objeto “Logger” se debe añadir la línea (en los desarrollos en los que se utilice esta clase “Logger” se recomienda crear este objeto a nivel global), para poder publicar los mensajes desde cualquier clase o función sin necesidad de volver a crear este objeto:

```
Logger lobjLogger = new Logger("","DEMO");
```

Posteriormente, y cada vez que se desee publicar un mensaje, utilizaremos la función “WriteMessageToLog”, que nos permitirá indicar el texto el mensaje, y la clasificación del mensaje (Error, Information, Warning).

```
lobjLogger.WriteMessageToLog("Error Message",
Logger.MessageClassification.Error);

lobjLogger.WriteMessageToLog("Information Message",
Logger.MessageClassification.Information);

lobjLogger.WriteMessageToLog("Warning Message",
Logger.MessageClassification.Warning);
```

En la ejecución de cada mensaje, esta función, creará un fichero por día donde almacenará los mensajes publicados con la clasificación del mensaje, y la fecha-hora exacta. Los nombres de los ficheros creados se iniciaran con el nombre de la aplicación pasado en el segundo parámetro del constructor de esta clase.

10.11 SwanCSharp.Miscellaneous

La clase Miscellaneous tiene como finalidad englobar todas aquellas funciones y procedimientos que no encajan en ninguna de las otras clases principales. Podemos definir esta clase como una clase donde se almacenan todas las funciones genéricas.

10.11.1 CalculationDiskSpace

La finalidad de procedimiento es el de calcular el espacio que queda libre en una unidad de disco dada (se calcula en Bytes).

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
long llngDiskSpaceInBytes = Miscellaneous.CalculateDiskSpace("c:")
```

10.11.2 ComputerCloseSession

La finalidad de procedimiento es el de cerrar la sesión abierta en el sistema operativo del ordenador donde se ejecuta la aplicación.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
Miscellaneous.ComputerCloseSession(10) // Ten seconds delay
```

10.11.3 Delay

La finalidad de procedimiento es el de producir un retardo de espera determinado por los segundos pasados en el único parámetro que recibe.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
Miscellaneous.Delay(10) // Ten seconds delay
```

10.11.4 ExistsLibrary

La finalidad de procedimiento es el de comprobar si una DLL o un fichero EXE existe en el sistema. En el primer parámetro se introduce el nombre de la librería a buscar sin indicar ruta concreta en disco.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
Miscellaneous.ExistsLibrary("kernel32.dll")
```

10.11.5 FormCentering

Este procedimiento se utiliza para centrar en pantalla un formulario creado y visible. Como parámetro espera recibir el formulario como objeto.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
Miscellaneous.FormCentering(pobjForm)
```

10.11.6 GetExecutionPath

Este procedimiento se utiliza para obtener la ruta completa de ejecución de nuestra aplicación. Se obtiene la ruta limpia, sin nombres de archivo, solamente la estructura completa de carpetas.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
string lstrApplicationPath = Miscellaneous.GetExecutionPath();
```

10.11.7 GetListOfCountries

Este procedimiento se utiliza para obtener la lista completa de países (en inglés o español). Al ejecutar la función se recibe por referencia dos arrays de strings, uno con la lista de países y otro con un código numérico para cada país, de tal forma que se recibe lo necesario para cargar un combo normal, o para cambiar un combo del interfaz gráfico personalizado "SwanCSharp_Controls".

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
string[] lstrCountriesCodes = new string[0];
string[] lstrCountriesNames = new string[0];
```

```
Miscellaneous.GetListofCountries(ref lstrCountriesCodes, ref
lstrCountriesNames, InterfaceLanguage.Spanish);
```

10.11.8 LoadDataInComboBox

Este procedimiento nos permite cargar en un ComboBox todos los elementos deseados, insertando en cada elemento los parámetros Text y Value deseados (algo que no permite directamente el objeto ComboBox). Para ello se va a crear un array de string con el parámetro "Text" de cada elemento, y otro array de "String" con el parámetro "Value" de cada elemento. En el tercer parámetro se pasa por referencia el "ComboBox" que deseamos cargar.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
private void Form1_Load(object sender, EventArgs e)
{
  string[] lstrData = new string[3];
  string[] lstrValue = new string[3];

  lstrData[0] = "Option 1";
  lstrValue[0] = "1";

  lstrData[1] = "Option 2";
  lstrValue[1] = "2";

  lstrData[2] = "Option 3";
  lstrValue[2] = "3";

   Miscellaneous.LoadDataInComboBox(lstrData, lstrValue, ref cmbTest);
}

private void cmbTest_SelectedIndexChanged(object sender, EventArgs e)
{
   Miscellaneous.ShowInformationMessage("Form1 ", "You've selected: " +
cmbTest.Text + " -- value: " + cmbTest.SelectedValue.ToString());
}
```

En el caso que el ComboBox a cargar sea el creado desde la clase SwanCSharp.WindowBase de SwanCSharp, vez de utilizar esta función (SwanCSharp.Miscellaneous.LoadDataInComboBox), habrá que utilizar la función paralela SwanCSharp_Controls.Miscellaneous.LoadDataInComboBox.

10.11.9 ObjectCentering

Este procedimiento se utiliza para centrar un control dentro de su "objeto padre". Como parámetro espera recibir el control como objeto.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
Miscellaneous.ObjectCentering(pobjObject)
```

10.11.10 RestartComputer

La finalidad de procedimiento es el de reiniciar el sistema operativo en el ordenador donde se ejecuta la aplicación.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
Miscellaneous.RestartComputer(10) // Ten seconds delay
```

10.11.11 ShowErrorMessage

La finalidad de procedimiento es el de mostrar una ventana de mensaje de error con el botón “OK” y el icono correspondiente. Como parámetros se pasan el titulo deseado para la ventana y el mensaje de la ventana.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
Miscellaneous.ShowErrorMessage(pstrTitle, pstrMessage)
```

Nota importante: En el caso de los formularios personalizados “SwanCSharp” creados desde el espacio de nombres “SwanCSharp_Controls”, será necesario utilizar la clase “WindowError” existente en ese espacio de nombres en lugar de esta función “ShowErrorMessage”.

10.11.12 ShowInformationMessage

La finalidad de procedimiento es el de mostrar una ventana de mensaje de información con el botón “OK” y el icono correspondiente. Como parámetros se pasan el titulo deseado para la ventana y el mensaje de la ventana.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
Miscellaneous.ShowInformationMessage(pstrTitle, pstrMessage)
```

Nota importante: En el caso de los formularios personalizados “SwanCSharp” creados desde el espacio de nombres “SwanCSharp_Controls”, será necesario utilizar la clase “WindowInformation” existente en ese espacio de nombres en lugar de esta función “ShowInformationMessage”.

10.11.13 ShowOKCancelQuestion

La finalidad de procedimiento es el de mostrar una ventana pregunta con el botón “OK” y “Cancel”, y el icono correspondiente. Como parámetros se pasan el titulo deseado para la ventana y el mensaje de la ventana. Devuelve un objeto “DialogResult” con la respuesta.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
Miscellaneous.ShowOKCancelQuestion(pstrTitle, pstrMessage)
```

Nota importante: En el caso de los formularios personalizados “SwanCSharp” creados desde el espacio de nombres “SwanCSharp_Controls”, será necesario utilizar la clase “WindowOKCancelQuestion” existente en ese espacio de nombres en lugar de esta función “ShowOKCancelQuestion”.

10.11.14 ShowWarningMessage

La finalidad de procedimiento es el de mostrar una ventana de mensaje de aviso con el botón “OK” y el icono correspondiente. Como parámetros se pasan el titulo deseado para la ventana y el mensaje de la ventana.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
Miscellaneous.ShowWarningMessage(pstrTitle, pstrMessage)
```

Nota importante: En el caso de los formularios personalizados “SwanCSharp” creados desde el espacio de nombres “SwanCSharp_Controls”, será necesario utilizar la clase “WindowWarning” existente en ese espacio de nombres en lugar de esta función “ShowWarningMessage”.

10.11.15 ShowYesNoQuestion

La finalidad de procedimiento es el de mostrar una ventana pregunta con el botón “Si” y “No”, y el icono correspondiente. Como parámetros se pasan el titulo deseado para la ventana y el mensaje de la ventana. Devuelve un objeto “DialogResult” con la respuesta.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
Miscellaneous.ShowYesNoQuestion(pstrTitle, pstrMessage)
```

Nota importante: En el caso de los formularios personalizados “SwanCSharp” creados desde el espacio de nombres “SwanCSharp_Controls”, será necesario utilizar la clase “WindowYesNoQuestion” existente en ese espacio de nombres en lugar de esta función “ShowYesNoQuestion”.

10.11.16 ShutDownComputer

La finalidad de procedimiento es el de apagar el ordenador donde se ejecuta la aplicación.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
Miscellaneous.ShutDownComputer(10) // Ten seconds delay
```

10.11.17 TextSlicingInLines

La finalidad de esta función es la de trocear un texto largo recibido en un “string” en la anchura de línea deseada. Mediante esta función se pueden encajar largos textos de párrafo en las líneas deseadas. Para realizar el corte de cada línea se calcula la longitud de cada palabra, para evitar que se corte por el medio, pasando la palabra entera a la siguiente línea.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
Miscellaneous.TextSlicingInLines(“long text”, 80) // Each line will have 80
characters width
```

10.12 SwanCSharp.Network

El espacio de nombres "SwanCSharp.Network" nos permite, mediante métodos y funciones, gestionar determinados aspectos de una red informática.

10.12.1 CheckITCPPortIsOpen

En la clase “Network” existe la función “CheckTCPPortIsOpen” que nos indica si un puerto TCP de una dirección o nombre de dominio dado está abierto o no.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
Boolean lblnResult = CheckTCPPortIsOpen("127.0.0.1", 1500);
```

10.12.2 GetIPNetworkData

En la clase “Network” existe la función “GetIPNetworkData” que devuelve toda la información asociada a las tarjetas de red Ethernet del sistema. La función devuelve un array de objeto “NetworkData” que contiene los campos Caption, description, MACAddress, DHCPEnabled, DHCPServer, DNSDomain, DNSHostName, IPAddress, IPSubnet, DefaultIPGateway, DNSServer, y SettingID.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
NetworkData[] lobjData = Network.GetIPNetworkData();

foreach (NetworkData lobjEthernet in lobjData)
{
   string lstrIPLan = lobjEthernet.IPAddress[0];
   string lstrMAC = lobjEthernet.MACAddress;
}
```

10.12.3 IsLocalIP

En la clase “Network” existe la función “IsLocalIP” que nos informa si una IP dada es de ámbito local o no.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
Boolean lblnIPLocal = Network.IsLocalIP(“255.255.255.255”);
```

10.12.4 IsValidIP

En la clase “Network” existe la función “IsValidIP” que nos informa si una IP dada tiene la estructura correcta y es válida.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
Boolean lblnIPValid = Network.IsValidIP(“255.255.255.255”);
```

10.13 SwanCSharp.Reporting

El espacio de nombres "SwanCSharp.Reporting" nos permite generar informes de datos para posteriormente visualizarlos y/o imprimirlos. Esta clase integra un ventana de visualización propia desde la cual se puede imprimir el informe.

Se decide utilizar archivos HTML para informes por su universalidad, se pueden ver en cualquier sistema, por el reducido espacio que ocupan, y por su flexibilidad.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Reporting;
```

Una línea de código de ejemplo puede ser:

```
DataConnectionAccess lobjConnection = new
DataConnectionAccess("Database.mdb", "", "", DatabaseManager.Access);
DataTable ldatData = lobjConnection.SQLSelectExecute("SELECT * FROM
Staff");
```

```
lobjConnection.CloseConnection();

GenerateHTMLReport lobjGenerate = new GenerateHTMLReport(ldatData,
lobjReportObjects, "Report", "", "Test of Report", "SwanCSharp.png", "",
SwanCSharp.Reporting.InterfaceLanguage.English,
SwanCSharp.Reporting.ReportViewWindowsSize.High, true, true);
```

El código anterior generará un informe utilizando los datos de ldatData, y con la definición que se haya pasado del objeto "ReportObjects". Dicho informe se guardará en la ruta especificada. Los parámetros a informar son:

* DataTable.- Los datos que se utilizarán para generar el informe.
* ReportObjects.- El objeto construido con la estructura básica del informe que va a permitir su construcción.
* Nombre del fichero.- El nombre del fichero que tendrá cuando se almacene en disco. El nombre del fichero se pasará sin extensión y SwanCSharp le añadirá la extensión ".html".
* Ruta del fichero.- La ruta del disco donde se va a crear el fichero html.
* Título del Informe.- El título que va a tener el informe.
* Nombre del fichero del logotipo.- Se puede pasar el nombre del fichero de imagen que se va a utilizar como logotipo a incluir en el informe.
* Ruta del fichero del logotipo.- La ruta en disco donde está el fichero de imagen del logotipo.
* Idioma del interfaz de pantalla.- Se elige el idioma en el que se va a mostrar la pantalla de visualización del informe.
* Tamaño de la ventana de visualización.- Se puede elegir entre Small, Medium, High.
* Fecha de creación del informe.- Si se pasa "true" en este parámetro, se imprimirá al final del informe la fecha y hora de creación.
* Visualización del informe.- Si se pasa "true" en este parámetro, una vez generado en el informe se mostrará una previsualización en pantalla de forma automática.

El generador de informes se ha creado con la idea de mostrar datos cuyo origen es un "DataTable" que se pude obtener de cualquier forma externa, o utilizando la clase DataAcess de SwanCSharp (como se puede ver en el ejemplo anterior). Aunque el informe permite en muchos apartados mostrar un texto "constante". La idea de "Reporting" es la de generar informes de datos concretos de un DataTable.

Para crear un informe es imprescindible utilizar la clase "ReportObjects", que es la clase que nos va a permitir personalizar la generación, permitiendo introducir datos constantes, establecer un texto determinado en negrita o subrayado, establecer una alineación del texto o de la información, colocar una barra de título, o colocar el detalle donde se va a mostrar los datos.

El primer paso para diseñar un informe con "Reporting" es dividir la informe en filas concretas, por ejemplo: un informe estándar se puede dividir en tres filas generales, una fila de cabecera, donde se va a mostrar datos generales, una fila de detalle donde se van a mostrar datos más concretos, y una última fila al pie del documento con datos de salida. En este documento vamos a mostrar el ejemplo de cómo se crea un informe que va a tener una primera fila con datos más generales, y una segunda fila con el detalle de la información concreta.

Para crear cada fila concreta podemos utilizar tres clases diferentes: ReportRowStandard, para crear una única fila con datos generales (los datos que se deseen),

ReportRowDoubleBox que es igual al anterior solo que nos permite trocear esa línea en dos partes bien diferenciadas y permitiendo configurar el ancho de cada una de las dos partes, y por último ReportRowDetail, que se utilizará para crear un detalle más general.

Vamos a mostrar un ejemplo concreto. Deseamos crear un informe que por cada registro de su base de datos (tabla Staff) vamos a mostrar una primera fila dividida en dos partes, la primera para mostrar únicamente el código del empleado, y la segunda para mostrar el resto de datos personales. Después existirá una segunda fila con el detalle del resto de campos.
Para la primera fila elegimos un objeto "ReportRowDoubleBox". Creamos el objeto de la siguiente forma.

```
ReportRowData[] lobjReportRowDataFirstBox = new ReportRowData[2];

lobjReportRowDataFirstBox[0].DataTableFieldName = "";
lobjReportRowDataFirstBox[0].BoldDataField = false;
lobjReportRowDataFirstBox[0].UnderlineDataField = false;
lobjReportRowDataFirstBox[0].PreviousConstantText = "Staff Data -- ";
lobjReportRowDataFirstBox[0].BoldPreviousConstant = false;
lobjReportRowDataFirstBox[0].UnderlinePreviousConstant = false;
lobjReportRowDataFirstBox[0].LaterConstantText = "";
lobjReportRowDataFirstBox[0].BoldLaterConstant = false;
lobjReportRowDataFirstBox[0].UnderlineLaterConstant = false;

lobjReportRowDataFirstBox[1].DataTableFieldName = "IDPerson";
lobjReportRowDataFirstBox[1].BoldDataField = true;
lobjReportRowDataFirstBox[1].UnderlineDataField = false;
lobjReportRowDataFirstBox[1].PreviousConstantText = "Person ID: ";
lobjReportRowDataFirstBox[1].BoldPreviousConstant = false;
lobjReportRowDataFirstBox[1].UnderlinePreviousConstant = false;
lobjReportRowDataFirstBox[1].LaterConstantText = "";
lobjReportRowDataFirstBox[1].BoldLaterConstant = false;
lobjReportRowDataFirstBox[1].UnderlineLaterConstant = false;

ReportRowData[] lobjReportRowDataSecondBox = new ReportRowData[2];

lobjReportRowDataSecondBox[0].DataTableFieldName = "Name";
lobjReportRowDataSecondBox[0].BoldDataField = true;
lobjReportRowDataSecondBox[0].UnderlineDataField = false;
lobjReportRowDataSecondBox[0].PreviousConstantText = "Name: ";
lobjReportRowDataSecondBox[0].BoldPreviousConstant = false;
lobjReportRowDataSecondBox[0].UnderlinePreviousConstant = false;
lobjReportRowDataSecondBox[0].LaterConstantText = " -- ";
lobjReportRowDataSecondBox[0].BoldLaterConstant = false;
lobjReportRowDataSecondBox[0].UnderlineLaterConstant = false;

lobjReportRowDataSecondBox[1].DataTableFieldName = "LastName";
lobjReportRowDataSecondBox[1].BoldDataField = true;
lobjReportRowDataSecondBox[1].UnderlineDataField = false;
lobjReportRowDataSecondBox[1].PreviousConstantText = "Last Name: ";
lobjReportRowDataSecondBox[1].BoldPreviousConstant = false;
lobjReportRowDataSecondBox[1].UnderlinePreviousConstant = false;
lobjReportRowDataSecondBox[1].LaterConstantText = "";
lobjReportRowDataSecondBox[1].BoldLaterConstant = false;
lobjReportRowDataSecondBox[1].UnderlineLaterConstant = false;
```

En el ejemplo anterior se puede comprobar cómo creamos dos objetos "ReportRowData" (que es el objeto básico de Reporting), uno para la primera parte de la fila, y otro para la

segunda parte de la fila. En cada parte podemos crear un array con tantos elementos como datos deseemos mostrar, en nuestro caso se muestran dos datos para la primera parte y dos datos para la segunda parte (pueden ser diferentes en cada parte).

Una vez creadas las dos partes las vamos a fusionar para crear la fila completa mediante el objeto ReportRowDoubleBox:

```
ReportRowDoubleBox lobjReportRowDoubleBox;
lobjReportRowDoubleBox.RowDataFirstBox = lobjReportRowDataFirstBox;
lobjReportRowDoubleBox.RowAlingmentFirstBox = ReportRowAlignment.Center;
lobjReportRowDoubleBox.ShowBorderFirstBox = true;
lobjReportRowDoubleBox.CellWidthFirstBox = 30;
lobjReportRowDoubleBox.RowDataSecondBox = lobjReportRowDataSecondBox;
lobjReportRowDoubleBox.RowAlingmentSecondBox = ReportRowAlignment.Left;
lobjReportRowDoubleBox.ShowBorderSecondBox = true;
```

En el objeto ReportRowDoubleBox se puede ver cómo le pasamos el objeto de cada parte, así como podemos pasar por separado para cada parte si se quiere mostrar el borde, o la alineación del texto. En este objeto también se determina el ancho que va a tener la primera parte de la caja, en este caso es 30, que quiere decir que el ancho de la primera parte será del 30%, por lo tanto el ancho de la segunda parte será automáticamente el 70%.

Cuando se pase este objeto al generador, lo que se va a crear en el informe será:

Staff Data -- Person ID: **1**	Name: **Forest** -- Last Name: **Nelson**

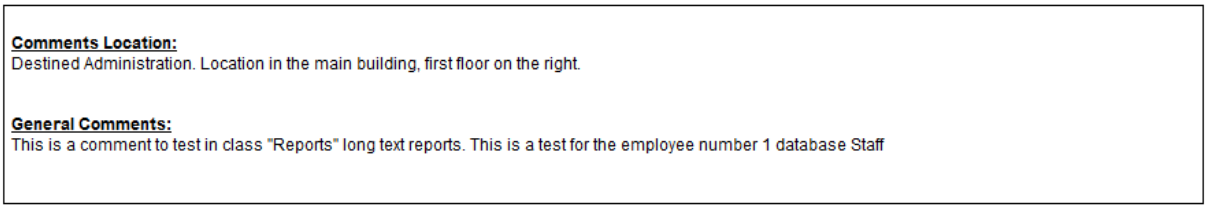
El siguiente paso es crear la fila de detalle, y para ello se utiliza la clase ReportRowDetail utilizando como base de cada dato la misma clase "ReportRowData" utilizada en el caso anterior:

```
ReportRowData[] lobjReportRowDetailData = new ReportRowData[2];

lobjReportRowDetailData[0].DataTableFieldName = "Comments";
lobjReportRowDetailData[0].BoldDataField = false;
lobjReportRowDetailData[0].UnderlineDataField = false;
lobjReportRowDetailData[0].PreviousConstantText = "Comments Location: ";
lobjReportRowDetailData[0].BoldPreviousConstant = true;
lobjReportRowDetailData[0].UnderlinePreviousConstant = true;
lobjReportRowDetailData[0].LaterConstantText = "";
lobjReportRowDetailData[0].BoldLaterConstant = false;
lobjReportRowDetailData[0].UnderlineLaterConstant = false;

lobjReportRowDetailData[1].DataTableFieldName = "Comments2";
lobjReportRowDetailData[1].BoldDataField = false;
lobjReportRowDetailData[1].UnderlineDataField = false;
lobjReportRowDetailData[1].PreviousConstantText = "General Comments: ";
lobjReportRowDetailData[1].BoldPreviousConstant = true;
lobjReportRowDetailData[1].UnderlinePreviousConstant = true;
lobjReportRowDetailData[1].LaterConstantText = "";
lobjReportRowDetailData[1].BoldLaterConstant = false;
lobjReportRowDetailData[1].UnderlineLaterConstant = false;
```

Como se puede comprobar en el código anterior vamos a mostrar dos campos en el detalle; ahora creamos el objeto ReportRowDetail:

```
ReportRowDetail lobjReportRowDetail;
lobjReportRowDetail.RowData = lobjReportRowDetailData;
lobjReportRowDetail.ShowBorder = true;
lobjReportRowDetail.RowAlingment = ReportRowAlignment.Left;
lobjReportRowDetail.RowDetailOrientation =
ReportDetailOrientation.Vertical;
```

Para este objeto de detalle hemos elegido la orientación vertical, alineación del texto izquierda y mostrar el borde. Cuando se pase este objeto al generador, lo que se va a crear en el informe será:



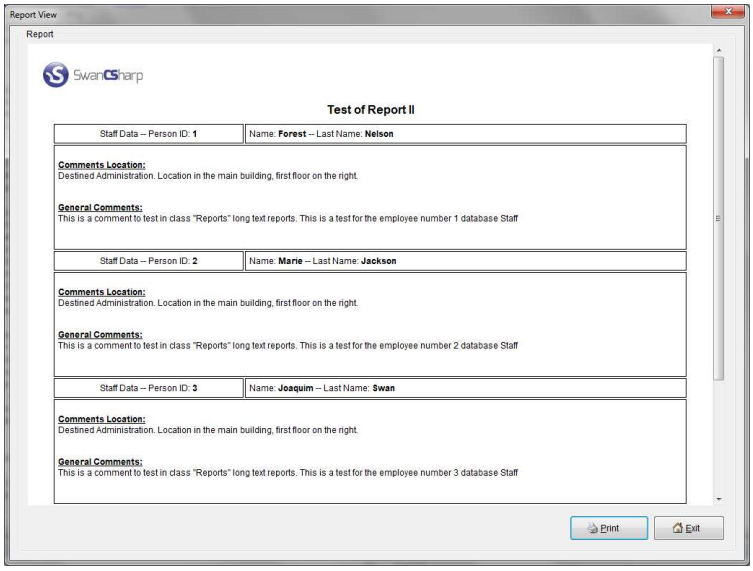
Resumiendo, desde la clase base ReportRowData hemos creado un informe que se divide en un primer objeto ReportRowDoubleBox, y un segundo objeto ReportRowDetail.
Pero para generar el informe hay que pasar un objeto ReportObjects. El siguiente paso es crear ReportObjects pasando cada uno de los dos objetos "Row" (se puede comprobar que podemos crear dentro ReportObjects tantos objetos fila como deseemos):

```
ReportObjects lobjReportObjects;
lobjReportObjects.ReportRows = new object[2];
lobjReportObjects.ReportRows[0] = lobjReportRowDoubleBox;
lobjReportObjects.ReportRows[1] = lobjReportRowDetail;
```

El último paso es llamar, pasando el objeto ReportObjects, a la clase que va a crear el fichero HTML con el informe a visualizar. Dicho fichero se podrá ver en cualquier navegador de Internet:

```
GenerateHTMLReport lobjGenerate = new GenerateHTMLReport(ldatData,
lobjReportObjects, "Report", "", "Test of Report II", "SwanCSharp.png", "",
SwanCSharp.Reporting.InterfaceLanguage.English,
SwanCSharp.Reporting.ReportViewWindowsSize.High, true, true);
```

La clase "GenerateHTMLReport" creará el informe "Report.html" y lo visualizará en pantalla de forma automática, mostrando la siguiente ventana:



En los ejemplos que acompañan a la librería (carpeta “Samples” que viene dentro del archivo ZIP SwanCSharp.zip descargado) existe un proyecto con ejemplos de "Reporting" cuyo nombre de proyecto es "Report".

10.13.1 ReportHTMLViewer

La clase ReportHTMLViewer nos permite mostrar en una ventana cualquier informe HTML ya existente, sin necesidad de volver a generarlo. Es necesario informar los siguientes parámetros:

* Nombre del fichero.- El nombre del fichero que tendrá cuando se almacene en disco.
* Ruta del fichero.- La ruta del disco donde se encuentra el fichero html.
* Título ventana.- El título que se va a mostrar en la ventana.
* Título del grupo.- El título que se va a mostrar en grupo de controles.
* Tamaño de la ventana de visualización.- Se puede elegir entre Small, Medium, High.
* Idioma del interfaz de pantalla.- Se elige el idioma en el que se va a mostrar la pantalla de visualización del informe.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Reporting;
```

Para crear el objeto cliente se utilizan los comandos:

```
ReportViewer.ReportHTMLViewer("report.html", "", "Report Saved", " Report",
ReportViewWindowSize.Medium,
SwanCSharp.Reporting.InterfaceLanguage.Spanish);
```

Nota importante: En el caso de los formularios personalizados “SwanCSharp” creados desde el espacio de nombres “SwanCSharp_Controls”, será necesario utilizar la clase “WindowReportView” existente en ese espacio de nombres en lugar de esta función “ReportHTMLViewer”.

10.14SwanCSharp.SNTP

La clase "SwanCSharp.SNTP" nos permite integrar en nuestras aplicaciones un actualizador de la fecha y hora del sistema. Existen muchos servidores de tiempo gratuitos (NTP Server, Network Time Protocol Server) que nos permiten obtener una fecha y hora actualizada y certificada. Con unas sencillas líneas de código podemos hacer que nuestras aplicaciones sincronicen (o actualicen) la fecha y la hora conectándose a un servidor NTP.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.SNTP;
```

Una línea de código de ejemplo puede ser:

```
SNTPClient lobjSNTPClient = new SNTPClient("IP or DNS of NTP Server");
lobjSNTPClient.Connect(5);
```

El constructor de la clase espera recibir la dirección DNS o la dirección IP del servidor NTP al que nos vamos a conectar. El método “Connect” espera recibir como parámetro el intervalo de fuera de tiempo (en segundos). Al ejecutar esas dos líneas de código nuestra aplicación se conectará al servidor NTP, recogerá la fecha y hora actualizada, y modificará el reloj del sistema con la nueva fecha y hora.

10.15SwanCSharp.Socket

Incorpora todas las funciones necesarias para gestionar sockets en aplicaciones desarrolladas en .NET Framework. La clase se divide en cuatro módulos, uno para ejecutar a modo de servidor para recibir todas las comunicaciones; y un segundo módulo a modo de cliente para enviar todas las órdenes al servidor. Después existe un tercer módulo para habilitar un servidor de archivos y un cuarto módulo para habilitar un cliente de archivos.

La clase para gestionar Sockets está pensada para desarrollar sistemas de transferencia de archivos, aplicaciones de Chat y comunicaciones, envío/recepción de imágenes vía TCP, aplicaciones de bases de datos mediante Cliente/Servidor, comunicación de máquina cliente a máquina servidor para realizar cambios de configuración en un ordenador núcleo de un proceso, etcétera, se pueden desarrollar miles de aplicaciones muy útiles de forma sencilla.

Por ejemplo, se puede desarrollar una aplicación para subir o descargar archivos alojados en un servidor mediante peticiones de múltiples clientes, sin necesidad de tener un servidor y un cliente FTP o SSH instalado en cada máquina, sin necesidad de depender de una estructura de comandos definida, y pudiendo personalizar todos los comandos.

También se puede utilizar para alojar un motor de grabación de datos con el soporte BBDD en un ordenador central a modo de servidor y grabar y solicitar la información desde cada cliente, esto nos evita desarrollar una página Web que no dispone de la flexibilidad de una aplicación de formularios de Windows.

La comunicación entre Servidor y Cliente socket de SwanCSharp está codificada para mayor seguridad y opera mediante la transmisión del conjunto “Comando” y “Dato” por lo tanto el Servidor socket no se puede utilizar con otro cliente socket ajeno a esta librería, y viceversa, el cliente no se puede utilizar en comunicación con un servidor socket que no sea el de esta librería.

Esta clase Socket se puede utilizar tanto la comunicación entre cliente y servidor vía LAN como vía WAN (Internet).

10.15.1 FileClient

La clase FileClient nos permite crear un cliente socket para enviar o recibir archivos a la clase FileServer.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Sockets;
```

Para crear el objeto cliente se utilizan los comandos:

```
FileClient lobjFileClient = new FileClient();
lobjFileClient.TransmissionResult += new
FileClient.TransmissionResultEventHandler(lobjFileClient_TransmissionResult
);
```

Para subir o bajar archivos del servidor se utiliza:

```
lobjFileClient.DownloadFile("test.txt", "", "192.168.0.15", 18000);
lobjFileClient.UploadFile("test.txt","","192.168.0.15", 18000);
```

Una vez enviado el mensaje, el evento `TransmissionResultEventHandler` salta cuando se recibe una respuesta del servidor a nuestro comando y a la transferencia solicitada. Al evento nos hemos suscrito en la segunda línea de la conexión cliente, por lo tanto necesitamos gestionar el evento añadiendo la función:

```
private static void lobjFileClient_TransmissionResult(string pstrFileName,
Boolean pblnDownloaded, Boolean pblnTransmissionResult)
{
}
```

Dentro de la función anterior recibimos el nombre del fichero objeto de la transacción, si la transacción es "Download" (true) o "Upload" (false), y el resultado de la transacción (true correcto, false error). Por ejemplo:

```
if (pblnTransmissionResult)
{
   if (pblnDownloaded)
   {
```

```
      Console.WriteLine("The file '" + pstrFileName + "' is downloaded
successfully");
   }
   else
   {
      Console.WriteLine("The file '" + pstrFileName + "' is uploaded
successfully");
   }
}
else
{
   if (pblnDownloaded)
   {
      Console.WriteLine("The file '" + pstrFileName + "' is downloaded
successfully");
   }
   else
   {
      Console.WriteLine("The file '" + pstrFileName + "' is uploaded
successfully");
   }
}
```

En los ejemplos que acompañan a la librería (carpeta “Samples” que viene dentro del archivo ZIP SwanCSharp.zip descargado) existe un ejemplo de un proyecto de Cliente de Ficheros que opera en combinación del proyecto de ejemplo de Servidor de Ficheros.

10.15.2 FileServer

La clase FileServer nos permite crear un servidor socket en un equipo informático que se va a utilizar para almacenar ficheros de cualquier tipo (servidor de archivos). El servidor de ficheros espera recibir órdenes del cliente de ficheros (FileClient) para subir o bajar un archivo determinado. La clase FileServer debe funcionar contra uno o varios equipos cliente que operen mediante la clase FileClient, por lo tanto existe una relación directa entre las clases FileServer y FileClient. Ambas clases nos permitirán transferir archivos sin necesidad de operar mediante protocolos estándar FTP, SSH, etcétera. Las clases FileServer y FileClient se pueden combinar con las clases SocketServer y SocketClient, para enviar por un canal TCP los comandos a ejecutar, y por este segundo canal los archivos a transferir, de tal forma que la comunicación más lenta de los archivos no va a interferir en la comunicación más rápida de los comandos y datos. El tamaño máximo de cada archivo a enviar no debe exceder de 20 Mb.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Sockets;
```

Para crear el objeto servidor y ponerlo en escucha se utilizan los comandos:

```
private static FileServer mobjServer;
mobjServer = new FileServer("192.168.0.15", 18000, "", false);
```

Pasando como primer parámetro la IP que tiene el ordenador en el que se va a ejecutar el servidor, pasando como segundo parámetro el puerto TCP deseado para la comunicación entre cliente y servidor. El tercer parámetro se utiliza para especificar si cada vez que el cliente descargue un fichero se desea que dicho fichero se deba de borrar en el servidor. Una vez ejecutadas las líneas anteriores, la aplicación permanecerá en escucha del puerto TCP elegido.

Cuando algún cliente de ficheros se conecte con el servidor, se ejecutará el evento "OpenClientConnectionEventHandler", por lo tanto deberemos declarar dicho evento y el procedimiento donde lo vamos a procesar:

```
mobjServer.OpenClientConnection += new
FileServer.OpenClientConnectionEventHandler(mobjServer_OpenClientConnection
);
```

Procedimiento donde se va a procesar:

```
private static void mobjServer_OpenClientConnection(string pstrIP, Int32
pintTCPPort)
{
   Console.WriteLine("Connection accepted with IP: " + pstrIP + " - TCP
Port: " + pintTCPPort.ToString());
}
```

Cada vez que un cliente envíe datos, mediante el evento "ReceiveDataEventHandler" se recibirán cinco parámetros: Fichero del proceso, IP Cliente, Puerto TCP Cliente, un valor booleano indicando si el archivo es descargado (true) o si el archivo es subido (false), y el Canal de comunicación).

Para suscribirse al evento se ejecuta:

```
mobjServer.ReceiveData += new
FileServer.ReceiveDataEventHandler(mobjServer_ReceiveData);
```

Para gestionar la trama recibida del cliente utilizamos:

```
private static void mobjServer_ReceiveData(string pstrFile, string pstrIP,
Int32 pintTCPPort, Boolean pblnDownload, object pobjChannel)
{
    if (pblnDownload)
    {
        Console.WriteLine("Requested download file " + pstrFile + " from
IP: " + pstrIP + " - TCP Port: " + pintTCPPort.ToString());
     }
     else
     {
        Console.WriteLine("Requested upload file " + pstrFile + " from IP:
" + pstrIP + " - TCP Port: " + pintTCPPort.ToString());
     }
}
```

En los ejemplos que acompañan a la librería (carpeta “Samples” que viene dentro del archivo ZIP SwanCSharp.zip descargado) existe un proyecto desarrollado de ejemplo de gestión de un servidor de ficheros.

Con las sencillas líneas de código descritas en este apartado tendremos un servidor de ficheros operativo que transmite la información aprovechando la máxima rapidez que nos permita la comunicación, ya sea en una red local LAN como en una red WAN.

10.15.3 SocketClient

La clase SocketClient nos permite crear un cliente socket para enviar órdenes, datos, e información a la clase SocketServer.
En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Sockets;
```

Para crear el objeto cliente se utilizan los comandos:

```
SocketClient lobjClient = new SocketClient("192.168.2.2", 5900);
lobjClient.ReceiveData+=new
SocketClient.ReceiveDataEventHandler(lobjClient_ReceiveData);
```

Para enviar un comando + datos al servidor se utiliza:

```
lobjClient.SendData("01", "abcdefg");
```

Para cerrar la conexión se utiliza:

```
lobjClient.CloseConnection();
```

Para enviar un comando y datos se recomienda crear la conexión cliente con el socket servidor, enviar el comando y los datos, recibir la respuesta, y cerrar posteriormente la conexión. El servidor Socket admite múltiples clientes conectados, pero no es conveniente utilizar una única conexión abierta para enviar varios comandos, es mejor crear la conexión enviar el comando y cerrarla para cada uno de los envíos.

Una vez enviado el mensaje, el evento `ReceiveDataEventHandler` salta cuando se recibe una respuesta del servidor a nuestro comando y a los datos enviados. Al evento nos hemos suscrito en la segunda línea de la conexión cliente, por lo tanto necesitamos gestionar el evento añadiendo la función:

```
private static void lobjClient_ReceiveData(string pstrCommand, string
pstrData)
{
}
```

Dentro de la función anterior recogemos la contestación al comando enviado al Server junto con los datos que nos envía, y realizamos los procesos que se estimen oportunos. Por ejemplo:

```
if (pstrCommand == "FF")
{
    Console.WriteLine("Command executed successfully");
}
else if (pstrCommand == "F0")
{
     Console.WriteLine("Command NOT executed successfully");
}
```

En los ejemplos que acompañan a la librería (carpeta “Samples” que viene dentro del archivo ZIP SwanCSharp.zip descargado) existe un ejemplo de un proyecto de Cliente Socket que opera en combinación del proyecto de ejemplo de Servidor Socket.

10.15.4 SocketServer

La clase SocketServer nos permite crear un servidor socket para recibir órdenes, datos, e información de la clase SocketClient.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Sockets;
```

Para crear el objeto servidor y ponerlo en escucha se utilizan los comandos:

```
private static SocketServer mobjServer;
mobjServer = new SocketServer("192.168.2.2", 8500);
```

Pasando como primer parámetro la IP que tiene el ordenador en el que se va a ejecutar el servidor, pasando como segundo parámetro el puerto TCP deseado para la comunicación entre cliente y servidor. Una vez ejecutadas las líneas anteriores, la aplicación permanecerá en escucha del puerto TCP elegido.

Cuando algún cliente socket se conecte con el servidor, se ejecutará el evento “OpenClientConnectionEventHandler”, por lo tanto deberemos declarar dicho evento y el procedimiento donde lo vamos a procesar:

```
mobjServer.OpenClientConnection += new
SocketServer.OpenClientConnectionEventHandler(mobjServer_OpenClientConnecti
on);
```

Procedimiento donde se va a procesar:

```
private static void mobjServer_OpenClientConnection(string pstrIP, Int32
pintTCPPort)
{
```

```
   Console.WriteLine("Connection accepted with IP: " + pstrIP + " - TCP
Port: " + pintTCPPort.ToString());
}
```

Cada vez que un cliente envíe datos, mediante el evento “ReceiveDataEventHandler” se recibirán cinco parámetros: Comando, Datos, IP Cliente, Puerto TCP Cliente, y el Canal de comunicación).

El Comando es un string de dos caracteres que coincide con un valor hexadecimal que va desde 00 hasta FF, permitiendo recibir 256 comandos diferentes. En el segundo parámetro se reciben los datos (un string sin límite). El tercer y cuarto parámetro recibimos la IP de cliente y el puerto TCP por el que hemos recibido la información. El último parámetro se utiliza para enviar una señal de confirmación o de error al cliente.

Para suscribirse al evento se ejecuta:

```
mobjServer.ReceiveData += new
SocketServer.ReceiveDataEventHandler(mobjServer_ReceiveData);
```

Para gestionar la trama recibida del cliente utilizamos:

```
private static void mobjServer_ReceiveData(string pstrCommand, string
pstrData, string pstrIP, Int32 pintTCPPort, object pobjChannel)
{
}
```

Dentro de la función anterior, cuando se quiera devolver un mensaje al cliente para indicar que el comando es correcto o incorrecto, por ejemplo, podemos enviar un comando “FF” cuando es correcto, y un comando “F0” cuando es incorrecto, para ello utilizamos el parámetro pobjChannel recibido:

```
mobjServer.SendResponse("FF", "Command '01' executed successfully",
pobjChannel);
```

```
mobjServer.SendResponse("F0", "Error: The command does not exist",
pobjChannel);
```

En los ejemplos que acompañan a la librería (carpeta “Samples” que viene dentro del archivo ZIP SwanCSharp.zip descargado) operamos con los comandos “01”, “02”, y “03” para realizar tres procesos diferentes, e utilizamos los comandos “FF” para confirmar éxito y “F0” para confirmar error. Pero el número de comandos a utilizar, y qué comandos se eligen es decisión de cada desarrollador, siempre debe ser un string de dos caracteres que coincida con un número hexadecimal válido, pero cada desarrollador puede elegir los que desee.

Con las sencillas líneas de código descritas en este apartado tendremos un servidor socket funcionando y con comunicación codificada, certificando que llega correctamente al

destinatario mediante con control de CRC del mensaje. Dependiendo de la imaginación se pueden desarrollar aplicaciones increíbles con este socket.

10.16 SwanCSharp.Users

El espacio de nombres “Users” nos permite gestionar todo lo relacionado con usuarios y perfiles. Con las clases definidas en este espacio de nombres podemos integrar en nuestras aplicaciones con muy pocas líneas de código y fácilmente una gestión completa de usuarios.

10.16.1 UserManagementSQLServer, UserManagementOracle, UserManagementFirebird, UserManagementMySQL y UserManagementAccess

El constructor de la clase “UserManagement” necesita utilizar una tabla de datos concreta ya sea en SQL Server, Oracle, Firebird, MySQL, o Access (una clase por cada gestor de base de datos). La finalidad de esta clase es la de incorporar a una aplicación una gestión de usuarios completa (incluidas ventanas de gestión). Por eso, cada vez que se construya el objeto “UserManagement”, esta clase comprobará si en nuestra base de datos existen las tablas “Profiles” y “Users” y si disponen de los campos necesarios. En caso contrario lo primero que hará de forma automática es crear toda la estructura necesaria en nuestra base de datos elegida (proceso que únicamente ejecutará la primera vez y sin intervención del desarrollador).

En el caso que el soporte de datos elegido sea SQL Server, los datos de la tabla “Users” serán cifrados por la clase “Management” para que nadie externamente a nuestros desarrollos pueda acceder a los datos de usuarios y sus contraseñas. Por lo tanto esta clase utiliza cifrado internamente para mayor seguridad.

En el caso de que la base de datos elegida sea Access, los datos no se cifran, pero se recomienda al usuario que proteja la base de datos mediante contraseña (desde la opción Seguridad de Microsoft Access).

En el caso de Oracle, MySQL, y Firebird los datos no se cifran, por lo tanto se recomienda utilizar los usuarios SQL que permiten ambos gestores de bases de datos.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Users;
```

Al construir el objeto de la clase UserManagement hay que recordar que en la base de datos pasada por parámetro, se crearán las tablas “Users” y “Profiles”. Los perfiles que se van a crear son: Superusuario, administrador, y usuario estándar. Con estos tres perfiles podremos jugar en nuestros desarrollos para establecer permisos.

Respecto a la tabla “Users” siempre se crea por defecto un usuario de partida con el perfil de “superusuario”, con el que podremos empezar a crear nuestros usuarios deseados. El usuario que se crea de partida es “**superadmin**” y la contraseña es “**sadmin**”.

También es importante destacar que las clases “UserManagement” se utilizarán a lo largo de todo el desarrollo de nuestra aplicación y por eso se recomienda crear el objeto mediante una variable global para que los datos del usuario activo, y las llamadas a las funciones de gestión, estén accesibles desde cualquier lugar de la aplicación. Para crear el objeto “UserManagement” se debe hacer:

```
public UserManagementSQLServer gobjUserManage = null;

private void Form1_Load(object sender, EventArgs e)
{
  gobjUserManage = new UserManagementSQLServer("PC-NAME\\SQL_INSTANCE",
"Database");

}
```

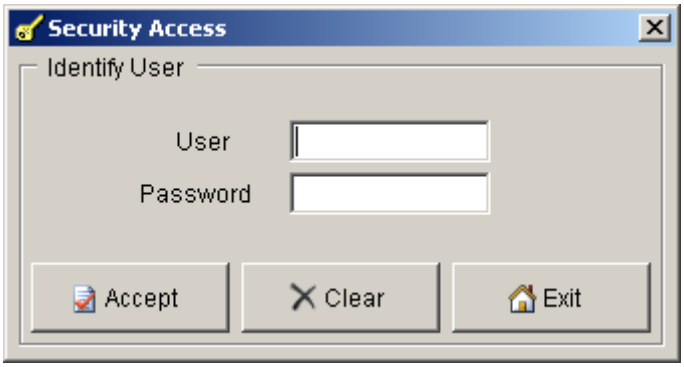
Para crear el objeto contra Oracle, Firebird, MySQL, o Access existen las clases “UserManagementOracle”, “UserManagementFirebird”, “UserManagementMySQL”, y “UserManagementAccess”.

10.16.1.1 ShowLoginWindow

Lo lógico es que al ejecutar la aplicación nos muestre una ventana de “login” para verificar la entrada con un usuario correcto. Por lo tanto antes de abrir nuestro formulario principal “Form1” deberemos llamar a la ventana de “Login”. Entonces el código fuente quedaría así:

```
public UserManagementSQLServer gobjUserManage = null;

private void Form1_Load(object sender, EventArgs e)
{
  gobjUserManage = new UserManagementSQLServer("PC-NAME\\SQL_INSTANCE",
"Database");
  gobjUserManage.ShowLoginWindow(InterfaceLanguage.English);
  if (!gobjUserManage.LoggedIn)
  {
     MessageBox.Show("Login incorrect.", "Test", MessageBoxButtons.OK,
MessageBoxIcon.Error);
     this.Dispose();
   }
}
```

Los formularios de Windows de esta clase se pueden mostrar en dos idiomas mediante el enumerado InterfaceLanguage, español e inglés. El código anterior mostrará una ventana de login, si el usuario introducido no es correcto, la aplicación se cerrará. A partir de la creación del objeto “User Management” disponemos de una variable global “gobjUserManage” disponible en toda la aplicación que tiene propiedades con los valores del usuario que ha accedido vía login (UserName, UserPassword, UserCompleteName, Profile).



Una vez que tenemos la variable pública con el objeto “UserManagement” podemos controlar, por ejemplo, que opciones de un MenuStrip de C# están habilitadas para el usuario activo según su perfil. Por ejemplo, según el siguiente código fuente:

```
private void MenuManage()
{
    // Standard User
    if (gobjUserManage.Profile == UserProfiles.StandardUser)
    {
        mnuSave.Enabled = false;
        mnuSetup.Enabled = false;
        mnuDelete.Enabled = false;
    }
    // Administrator
    else if (gobjUserManage.Profile == UserProfiles.Administrator)
    {
        mnuDelete.Enabled = false;
        mnuAddUser.Enabled = false;
    }
}
```

Si el usuario activo “logueado” tiene el perfil de “Super Usuario”, tiene acceso a todos los menús de opciones. Si el usuario dispone del perfil “administrador”, no tendrá acceso al menú “Delete” y al menú “Add User”. Si el usuario dispone de un perfil “StandardUser” no tiene acceso a los menús “Save”, “Setup”, y “Delete”. La función “MenuManage” se ejecuta justo después de confirmar que el acceso login es correcto, y se establecerán los permisos de menú según su perfil.

Nota importante: En el caso de los formularios personalizados “SwanCSharp” creados desde el espacio de nombres “SwanCSharp_Controls”, será necesario utilizar la clase “WindowLoginUser” existente en ese espacio de nombres en lugar de esta función “ShowLoginWindow”.

10.16.1.2 ShowUserAddWindow

Además de la ventana de “login” existe la ventana “Add User” que nos permitirá incorporar una opción para dar de alta nuevos usuarios. Un ejemplo de llamada a la ventana “Add User” es:

```
private void mnuAddUser_Click(object sender, EventArgs e)
```

```
{
   gobjUserManage.ShowUserAddWindow(gobjUserManage.Profile,
InterfaceLanguage.English);
}
```

Como primer parámetro se pasa el perfil del usuario activo en cada instante, y como segundo parámetro se pasa el idioma de la interfaz gráfica. Un ejemplo de pantalla es:

Esta pantalla nunca va a permitir a un usuario con perfil “StandarUser” crear otro usuario. A un usuario con perfil “Administrator” solo le permite crear otro usuario “administrador” u otro usuario “StandarUser”. Para el usuario “SuperAdmin” no existen restricciones de ningún tipo.

Nota importante: En el caso de los formularios personalizados “SwanCSharp” creados desde el espacio de nombres “SwanCSharp_Controls”, será necesario utilizar la clase “WindowAddUser” existente en ese espacio de nombres en lugar de esta función “ShowUserAddWindow”.

10.16.1.3 ShowUserPasswordChangeWindow

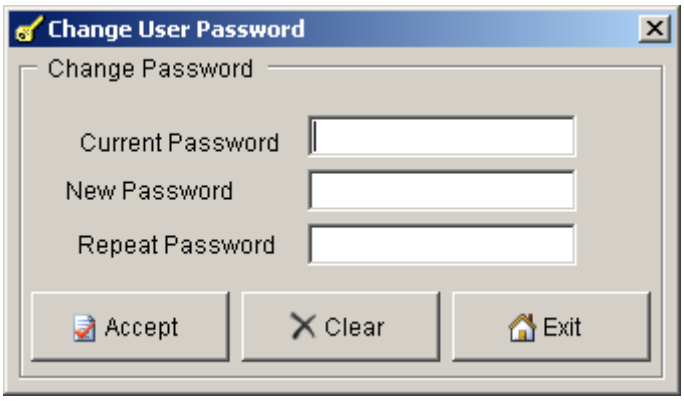
La ventana de “Password Change” nos permitirá modificar la contraseña del usuario activo que está logueado en ese momento. Este método abrirá una ventana donde se introducirá la contraseña vigente, y se introducirá la nueva contraseña deseada. Al pulsar en “Aceptar” la contraseña será modificada siempre y cuando la contraseña actual coincida con la introducida en pantalla.

Para llamar a la ventana de cambio de contraseña se escribe el siguiente código:

```
try
{
    gobjUserManage.ShowUserPasswordChangeWindow(gobjUserManage.UserName,
InterfaceLanguage.English);
}
catch (Exception ex)
{
```

```
    MessageBox.Show(ex.Message);
}
```

Como primer parámetro se pasa el usuario activo en cada instante, y como segundo parámetro se pasa el idioma de la interfaz gráfica. Un ejemplo de pantalla es:



Nota importante: En el caso de los formularios personalizados “SwanCSharp” creados desde el espacio de nombres “SwanCSharp_Controls”, será necesario utilizar la clase “WindowPasswordUser” existente en ese espacio de nombres en lugar de esta función “ShowUserPasswordChangeWindow”.

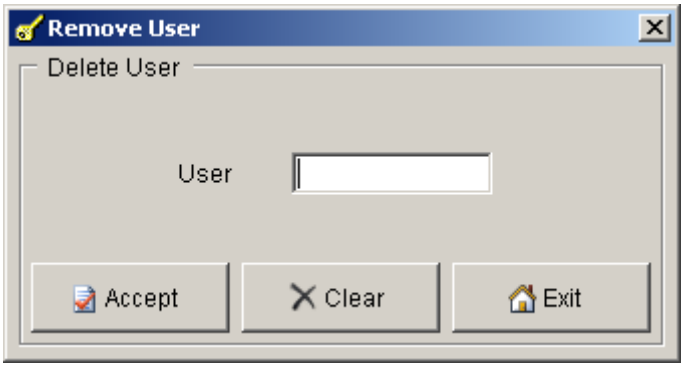
10.16.1.4 ShowUserRemoveWindow

La ventana de “Remove User” nos permitirá eliminar usuarios de la base de datos. Para eliminar a un usuario únicamente es necesario introducir su nombre. Es importante saber que existen unas reglas para poder borrar usuarios: Si el usuario activo es “SuperUser” puede eliminar a cualquier otro usuario sea cual sea su perfil; si el perfil del usuario activo es “Administrador”, puede eliminar únicamente a usuarios “Standard”. El usuario “Standard” no tiene permisos para borrar a ningún otro usuario sea cual sea su perfil.

Para llamar a la ventana de borrado de usuarios se escribe el siguiente código:

```
try
{
   gobjUserManage.ShowUserRemoveWindow(gobjUserManage.Profile,
InterfaceLanguage.English);
}
catch (Exception ex)
{
   MessageBox.Show(ex.Message);
}
```

Como primer parámetro se pasa el perfil del usuario activo en cada instante, y como segundo parámetro se pasa el idioma de la interfaz gráfica. Un ejemplo de pantalla es:



Nota importante: En el caso de los formularios personalizados “SwanCSharp” creados desde el espacio de nombres “SwanCSharp_Controls”, será necesario utilizar la clase “WindowRemoveUser” existente en ese espacio de nombres en lugar de esta función “ShowUserRemoveWindow”.

10.16.1.5 ChangeUserPassword

Con la función “ChangeUserPassword” podemos, desde código fuente, cambiar la contraseña de un usuario activo.

```
ChangeUserPassword(gobjUserManage.UserName,”Current Password”, “New
password”)
```

10.16.1.6 IsValidUser

Con la función “IsValidUser” podemos, desde código fuente (la ventana de login ya la utiliza automáticamente), comprobar cuando lo deseemos si un usuario determinado es válido. Esta función se utiliza para comprobar si un usuario y contraseña existen en la tabla de usuarios. Además de devolver “true” en caso de que exista, y “false” en caso de que no exista, devuelve por referencia el nombre completo del usuario y el perfil de dicho usuario (si existe).

```
if (IsValidUser(pstrUser, pstrUserPassword, ref pstrUserCompleteName, ref
pintProfile))
{
   lblnLoggedIn = true;
}
else
{
   lblnLoggedIn = false;
}
```

10.16.1.7 ExistsUserName

Dado un nombre de usuario, la función ExistsUserName nos comprueba si ese nombre de usuario existe en la tabla “Users”. Está comprobación la hace automáticamente la ventana “User Add” pero en un momento dado, vía código fuente, nos puede interesar, como paso previo a dar de alta, comprobar si un usuario existe o no.

```
if (ExistsUserName(pstrUser))
{
    throw new ApplicationException("Can not create the user name because
already exists");
}
else
{
    UserAdd(mstrUser, mstrUserPassword, mstrUserCompleteName,
UserProfiles.UserProfile);
}
```

10.16.1.8 ProfileUserName

La función “ProfileUserName” nos permite obtener el perfil de un usuario pasado por parámetro.

```
UserProfiles lenuProfile = ProfileUserName(lstrUser);
```

10.16.1.9 UserAdd

La función “UserAdd” nos permite crear un usuario nuevo vía código fuente, sin necesidad de utilizar la ventana de creación de usuarios.

```
UserAdd(pstrUser, pstrUserPassword, pstrUserCompleteName,
UserProfiles.UserProfile);
```

10.16.1.10 UserRemove

La función “UserRemove” nos permite eliminar un usuario determinado por parámetro. Para que el borrado sea ejecutado se tienen que cumplir las normas de borrado especificadas en este documento en el apartado de “UserManagement”.

```
UserRemove(string pstrUserName, UserProfiles penuProfileActive,
UserProfiles penuProfileToDelete);
```

10.17SwanCSharp.Validations

Incorpora todas las funciones necesarias para realizar rápidas validaciones de datos en aplicaciones desarrolladas en .NET Framework.

10.17.1 FileNameWithPathValidate

La finalidad de esta función es la de validar si un valor string pasado por parámetro es una ruta completa válida de un fichero. La función realiza las siguientes validaciones:

- Comprueba que el fichero existe. En caso contrario devuelve false.

- Comprueba que en la definición del nombre de fichero se incluye la ruta. En caso contrario devuelve false.
- Del nombre de fichero extrae la ruta y comprueba que existe. En caso contrario devuelve false.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
string lstrResult = "";
if (Validations.FileNameWithPathValidate(pstrFileNameWithPath))
{
   strResult = "The path and the filename are valid";
}
else
{
   strResult = "The path and the filename are NOT valid";
}
```

10.17.2 HexadecimalInString

La finalidad de esta función es la de validar si una cadena “string” se compone de una cadena hexadecimal válida.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
if (!Validations.HexadecimalInString(pstrCommand))
{
   throw new ApplicationException("The Command parameter must be a valid
hexadecimal string");
}
```

10.17.3 HigherNumber

La finalidad de esta función es la de devolver el número más grande de dos números pasados por parámetro.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
int lintHighNumber =  Validations.HigherNumber(10,15)
```

10.17.4 IsNumeric

La finalidad de esta función es verificar si una cadena string (o char) dada es un valor numérico o no. La función tiene cuatro sobrecargas que permiten pasar desde un único parámetro hasta tres parámetros, pudiendo pasar el número como string, como char, pudiendo especificar cuál es el separador decimal o el número de decimales.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
string lstrNumber = “215,67”;
Boolean lblnIsNumeric =  Validations.IsNumeric(lstrNumber, “,”, 2)
```

10.17.5 IsValidDate

La finalidad de esta función es la de comprobar si una fecha recibida (en tipo de datos “string”) es una fecha válida en calendario y formato.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
string lstrDate = "20/02/2013";
if (IsValidDate(lstrDate))
{
   Console.WriteLine("The date is valid");
}
else
{
   Console.WriteLine("The date is NOT valid");
}
```

10.17.6 IsValidEmail

La finalidad de esta función es la de comprobar si un email dado dispone del formato correcto para una dirección de correo electrónico.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
string lstrEmail = "test@swancsharp.com";
if (IsValidEmail(lstrEmail))
{
   Console.WriteLine("The email is valid");
}
   else
{
   Console.WriteLine("The email is NOT valid");
}
```

10.17.7 LowerNumber

La finalidad de esta función es la de devolver el número más pequeño de dos números pasados por parámetro.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp;
```

Un código de ejemplo puede ser:

```
int lintLowerNumber =  Validations.LowerNumber(10,15);
```

10.18SwanCSharp.Video

El espacio de nombres “Video” nos permite obtener fácilmente un stream de vídeo generado por una cámara IP con soporte HTTP. Las cámaras IP que permiten recoger el stream de video mediante http suelen tener “programas” grabados (CGI) que permiten extraer el indistintamente el vídeo (normalmente MJPEG) o imágenes sueltas (normalmente JPEG).

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp.Video;
```

El espacio de nombres “Video” contiene una clase principal llamada Streaming con dos constructores: el primero solo hay que indicar la dirección http del CGI que nos devuelve cada JPEG; el segundo además de la dirección http nos permite personalizar el intervalo (en milisegundos) con el que deseamos descargar cada frame de imagen.

Por ejemplo, si pasamos únicamente la dirección http cada frame se descargará en intervalos de 10 milisegundos, y el constructor sería el siguiente:

```
Streaming mobjStreaming = new Streaming("http://192.168.x.x /axis-
cgi/jpg/image.cgi");
```

En el caso anterior estamos estableciendo la dirección http de descarga de cada frame de una cámara AXIS. Cada fabricante de cámara tendrá su propia dirección http.

Si deseamos personalizar el tiempo de descarga de cada frame, por ejemplo que descargue un frame cada 100 milisegundos (10 fps), sería:

```
Streaming mobjStreaming = new Streaming("http://192.168.x.x /axis-
cgi/jpg/image.cgi", 100);
```

Si la cámara se ha establecido autenticación mediante usuario y contraseña, hay que añadir a continuación lo siguiente:

```
mobjStreaming.User = "user";
mobjStreaming.Password = "password";
```

El último paso es establecer el evento que saltará con la llegada de cada frame e iniciar la captura de frames:

```
mobjStreaming.NewFrame += new
Streaming.NewFrameEventHandler(mobjStreaming_NewFrame);

mobjStreaming.StartCapture();
```

Solo nos queda crear el método del evento “NewFrame”, el cual saltará con cada frame recogido llegando por parámetro en forma de un objeto “Bitmap”:

```
private void mobjStreaming_NewFrame(Bitmap pobjFotograma)
{
     /*
      * Code for "pobjFotograma" image management here
      */
}
```

Si en algún momento se desea detener la captura de fotogramas, se puede utilizar el siguiente comando:

```
mobjStreaming.StopCapture();
```

11. REFERENCIA DE USO (SwanCSharp_Controls)

A continuación se va a describir la referencia de uso de cada una de las funciones incorporadas a la librería en el espacio de nombres SwanCSharp_Controls. Las referencias se agrupan por las clases a las que pertenecen en el orden lógico de su desarrollo.

11.1 SwanCSharp_Controls.WindowBase

La clase "WindowBase" consta de un objeto heredable que nos permite en nuestros desarrollos romper con la interfaz gráfica de ventanas que nos proporciona el sistema

operativo y operar con otra interfaz diferente, ampliando las posibilidades de personalización limitadas en un desarrollo estándar.

Por ejemplo, en un desarrollo estándar se puede ocultar por separado el botón de minimizar y el de maximizar, o podemos deshabilitar todos los botones (ControlBox) de una vez; pero no podemos ocultar solo el de cerrar, y si ocultamos todos (ControlBox), no se puede incluir un icono gráfico en la barra de título de la ventana. Con la clase WindowBase de SwanCSharp podemos ocultar por separado cada botón, además de incluir el icono a la ventana en cualquier caso, incluso podemos alterar el orden de los botones cerrar, maximizar/restaurar, y minimizar.

También podemos cambiar el color del texto de la barra de título, algo que no se puede hacer en la interfaz estándar, así como podemos bloquear la posibilidad de mover la ventana (Moveable) característica retirada desde Visual Basic 6.0.

La clase WindowBase habilita la posibilidad de incluir transparencia de fondo para las ventanas y un montón de funciones que se pasan a describir en párrafos siguientes.

El entorno gráfico "SwanCSharp_Controls" incluye seis temas gráficos diferentes a elegir, basándose cada tema en un color central concreto (azul, gris, negro, oro, rojo, y verde).

Para poder utilizar el espacio de nombres "SwanCSharp_Controls" es necesario tener instalado en el equipo la versión **3.0 de .Net Framework** o **superior**.

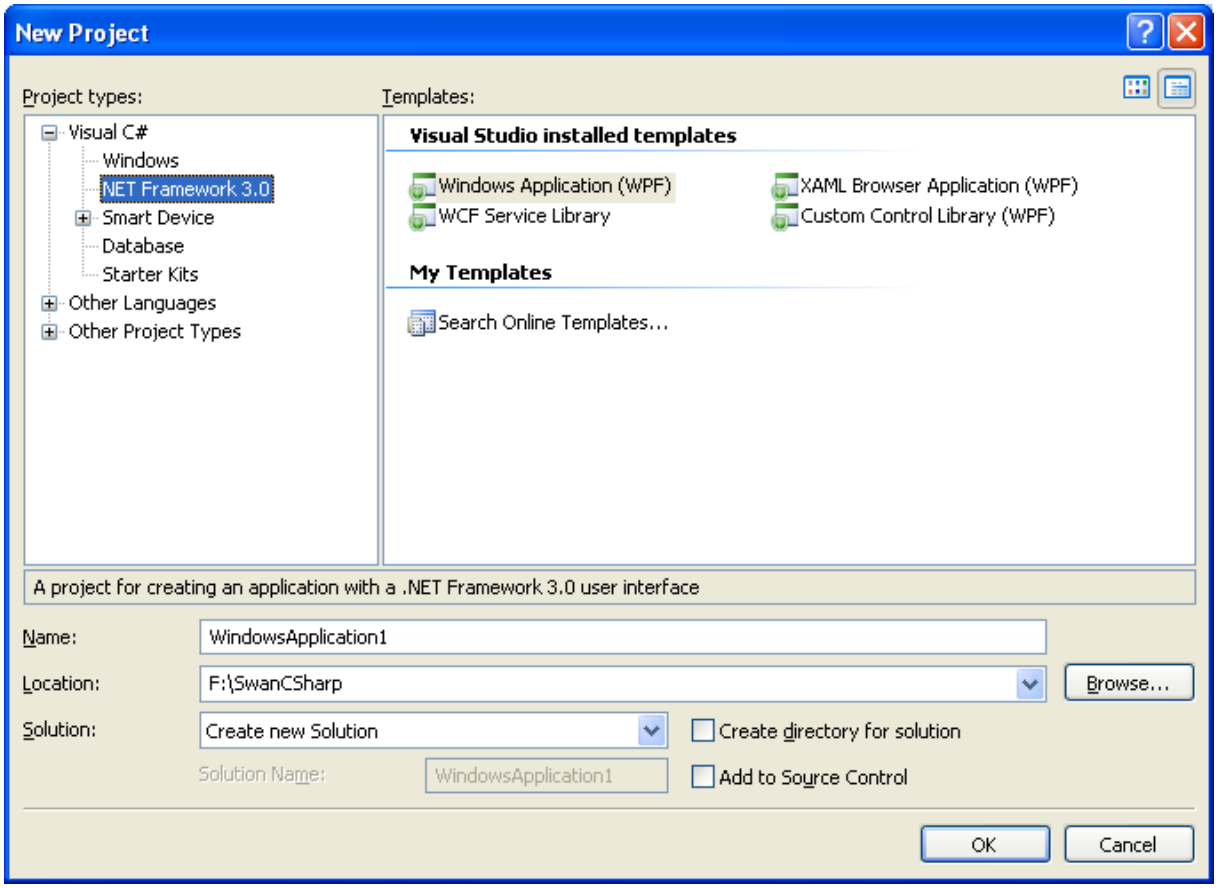
Respecto al entorno IDE de desarrollo, se puede utilizar sin problemas el Visual Studio 2008 o superior. Respecto al Visual Studio 2005 se tiene que instalar el módulo "**Visual Studio 2005 extensions for .NET Framework 3.0 (WPF)**", que se puede descargar gratuitamente de Internet. Sin la instalación de este módulo no se puede desarrollar utilizando el espacio de nombres SwanCSharp _Controls.

Los ejemplos mostrados en este documento están ejecutados bajo la versión Visual Studio 2005 con "**Visual Studio 2005 extensions for .NET Framework 3.0 (WPF)**" instalado.

Para mostrar un funcionamiento general de la clase WindowBase se ha creado un proyecto de ejemplo llamado “SwanWindow” que muestra las características gráficas más importantes de esta clase. Al ejecutar el ejemplo pedirá un usuario y contraseña que en ambos casos es “admin”.

11.1.1 Creación de un proyecto nuevo

Para crear un nuevo proyecto que pueda trabajar con el nuevo estilo de ventanas y controles es necesario ir a la opción de menú Archivo -> Nuevo Proyecto. En la ventana de aparecerá en pantalla hay que seleccionar (a la izquierda) dentro de “Visual C#” un proyecto “.NET Framework 3.0”, y posteriormente (a la derecha) “Windows Application (WPF)”. Se selecciona el nombre deseado al proyecto y la carpeta donde se va almacenar y se pulsa en el botón “OK”.



11.1.2 Crear una Ventana "SwanCSharp"

Al crear un nuevo proyecto WPF se creará un archivo inicial App.xaml que ejecutará una ventana inicial llamada Window1.xaml. Para poder crear una ventana "SwanCSharp" es necesario referenciar al proyecto el archivo "SwanCSharp.dll". También es necesario referenciar al proyecto la clase "System.Drawing" de .Net Framework.

Una vez referenciada la librería SwanCSharp, hay que cambiar la ventana Window1.xaml para que no herede de la clase estándar Window, sino que herede de la clase WindowBase del espacio de nombres SwanCSharp_Controls existente en la librería SwanCSharp.dll. Se cambia la herencia de la siguiente forma:

- Se eliminan de Window1.xaml las etiquetas `<Grid></Grid>`.
- En Window1.xaml se cambia el nombre de etiqueta `<Window x:Class="WindowsApplication1.Window1"` por `<src:WindowBase x:Class="Pruebas2.Window1"`.
- Se cambia el nombre de etiqueta `</Window>` por `</src:WindowBase>`.
- Debajo de la línea `xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"` se añade la nueva línea con la referencia de la clase WindowBase que es `xmlns:src="clr-namespace:SwanCSharp_Controls;assembly=SwanCSharp"`.
- En el archivo Window1.xaml.cs se añade la referencia al espacio de nombres SwanCSharp incluyendo la línea "`using SwanCSharp_Controls;`".

- En el archivo Window1.xaml.cs se cambia la línea "`public partial class Window1 : System.Windows.Window`" por "`public partial class Window1 : WindowBase`". De esta forma le indicamos que vamos a heredar de la clase WindowBase.
- En el constructor principal de la clase (`public Window1()`) hay que añadirle seguidamente "`base(true)`", quedando el constructor "`public Window1() : base(true)`". El valor "`true`" activa la transparencia de fondo para nuestra ventana, si no deseamos dicha transparencia cambiamos a "`false`". En esta sobrecarga del constructor el color central del interfaz (tema) siempre será el azul.
- Existe una segunda sobrecarga en el constructor principal de la clase (`public Window1()`) que nos permite seleccionar el color central de la aplicación (tema) además de seleccionar la transparencia. Por ejemplo: "`base(WindowTheme.Black, false)`". Existen los siguientes temas: `WindowTheme.Black`, `WindowTheme.Blue`, `WindowTheme.Gold`, `WindowTheme.Gray`, `WindowTheme.Green`, `WindowTheme.Red`.

Una vez hechos los cambios obtenemos del archivo original Window1.xaml:

```
<Window x:Class="WindowsApplication1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="WindowsApplication1" Height="300" Width="300"
    >
    <Grid>
    </Grid>
</Window>
```

El archivo modificado Window1.xaml:

```
<src:WindowBase x:Class="WindowsApplication1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:src="clr-namespace:SwanCSharp_Controls;assembly=SwanCSharp"
    Title="WindowsApplication1" Height="300" Width="300"
    >
</src:WindowBase>
```

También obtenemos del archivo original Window1.xaml.cs:

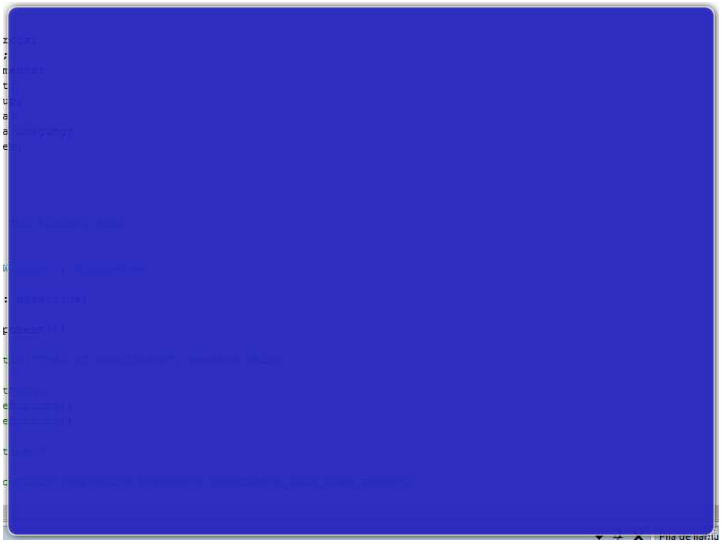
```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
```

```
namespace WindowsApplication1
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>

    public partial class Window1 : System.Windows.Window
    {

        public Window1()
        {
            InitializeComponent();
        }

    }
}
```

El archivo modificado Window1.xaml.cs:

```
using SwanCSharp_Controls;
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;


namespace WindowsApplication1
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>

    public partial class Window1 : WindowBase
    {

        public Window1() : base(WindowTheme.Blue, true)
        {
            InitializeComponent();
        }

    }
}
```

Al ejecutar el proyecto WindowsApplication1 nos mostrará en pantalla la ventana de estilo “SwanCSharp” que incluye transparencia sobre el fondo y múltiples ventajas que se irán explicando a lo largo de este documento.



Para el tema `WindowTheme.Black`:

```
namespace WindowsApplication1
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>

    public partial class Window1 : WindowBase
    {

        public Window1() : base(WindowTheme.Black, true)
        {
            InitializeComponent();
        }

    }
}
```

Para el tema `WindowTheme.Gold`:

```
namespace WindowsApplication1
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>

    public partial class Window1 : WindowBase
    {

        public Window1() : base(WindowTheme.Gold, true)
        {
            InitializeComponent();
        }

    }
}
```

Para el tema `WindowTheme.Gray`:

```
namespace WindowsApplication1
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>

    public partial class Window1 : WindowBase
    {

        public Window1() : base(WindowTheme.Gray, true)
        {
            InitializeComponent();
        }
```

```
    }
}
```

Para el tema `WindowTheme.Green`:

```
namespace WindowsApplication1
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>

    public partial class Window1 : WindowBase
    {

        public Window1() : base(WindowTheme.Green, true)
        {
            InitializeComponent();
        }

    }
}
```

Para el tema `WindowTheme.Red`:

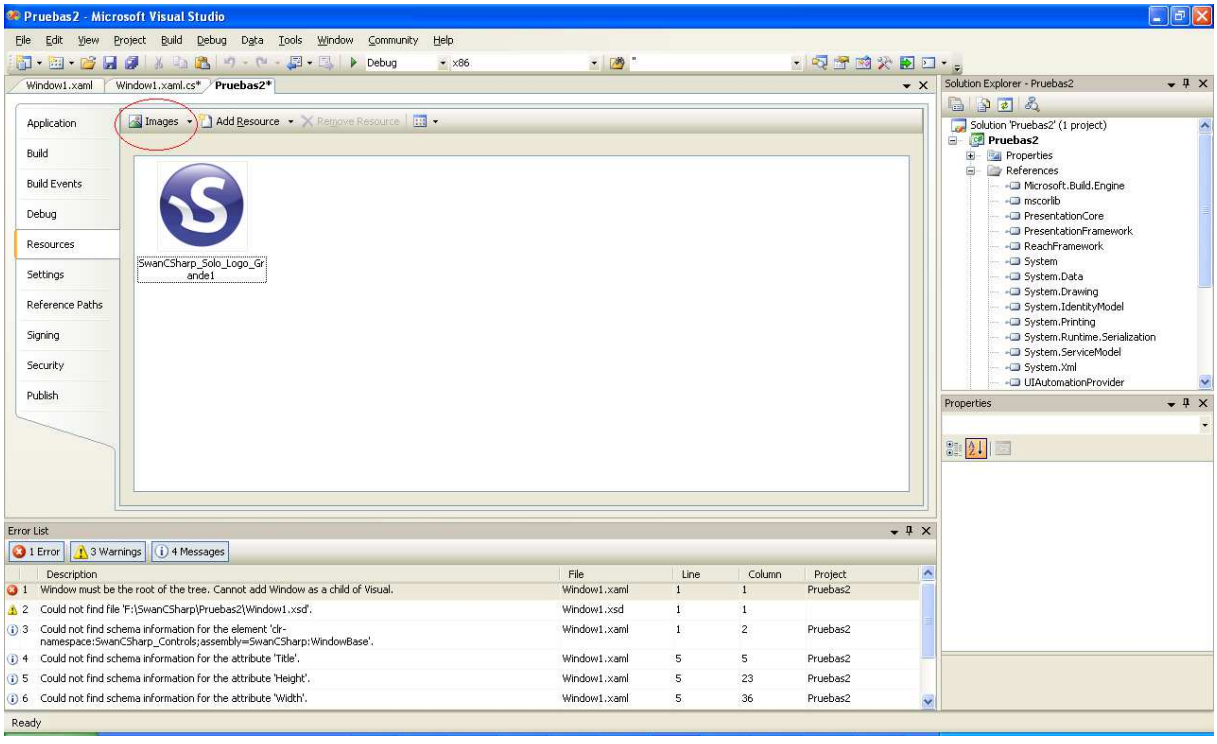
```
namespace WindowsApplication1
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>

    public partial class Window1 : WindowBase
    {

        public Window1() : base(WindowTheme.Red, true)
        {
            InitializeComponent();
        }

    }
}
```

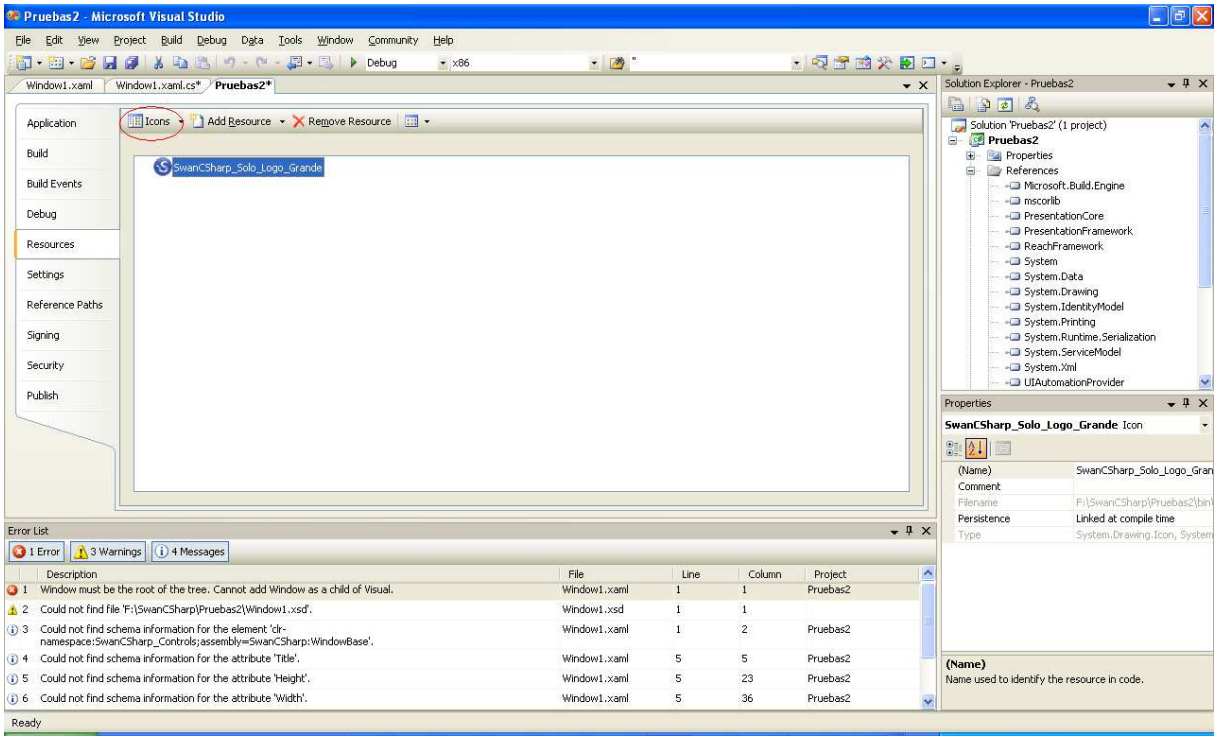
Nota Importante: En las versiones de Windows Presentation Foundation (WPF) incluidas en el .NET Framework, hasta la fecha de edición de este documento, **NO PERMITEN** heredar ventanas WPF porque los archivos XAML no son heredables. En el caso de SwanCSharp se consigue una herencia correcta, aunque el formulario **NUNCA** se podrá previsualizar en el IDE y los objetos sobre él se tendrán que colocar mediante los métodos descritos más abajo en este documento.

11.1.3 Imágenes e iconos en las ventanas "SwanCSharp"

Muchos métodos del espacio de nombres "SwanCSharp_Controls" permiten utilizar imágenes y/o iconos, y estos archivos se recomiendan que se integren en nuestros proyectos de desarrollo mediante el archivo de recursos del proyecto Visual Studio. De esta forma se integrarán con nuestros compilados y podrán ser llamados fácilmente desde el código para ser pasados por parámetro.

En la imagen inferior se pueden ver dos pantallas de ejemplo: La primera donde se integran imágenes en el archivo de recursos, y la segunda donde se integran los iconos en el archivo de recursos.





11.1.4 Propiedad “MainGrid”

En la clase “WindowBase” existe una propiedad de tipo “Grid” llamada “MainGrid” que va a tener una gran importancia en el desarrollo de una ventana “SwanCSharp”: Por medio de la propiedad “MainGrid” vamos a poder colocar todos los controles y objetos a utilizar en cada formulario.

11.1.5 Propiedades iniciales de la Ventana “SwanCSharp”

Existen una serie de propiedades iniciales que se pueden establecer en una ventana “SwanCSharp”, dichas propiedades iniciales determinarán la visualización gráfica de la ventana.

11.1.5.1 Título de la ventana

El título de la ventana se establecerá mediante el método SetWindowTitle de la clase WindowBase y dentro del constructor de la ventana XAML creada (en nuestro ejemplo Window1.xaml.cs). Como primer parámetro se pasa el texto que se va a mostrar como título de la ventana; en el segundo parámetro se informa del color deseado para el texto, característica que no se puede aplicar en un WinForm estándar. Un ejemplo puede ser:

```
public Window1() : base(false)
{
     InitializeComponent();
     SetWindowTitle("Test of SwanCSharp", Brushes.White);
}
```

Con la línea “SetWindowTitle” incluida debajo de “InitializeComponent” la ventana que se mostrará al ejecutar la solución será la siguiente:

11.1.5.2 Icono de la ventana

El icono de la ventana se establecerá mediante el método SetIconWindowTitle de la clase WindowBase y dentro del constructor de la ventana XAML creada (en nuestro ejemplo Window1.xaml.cs). Existen dos sobrecargas para este método, en la primera se pasa como parámetro un objeto “Bitmap”, y en la segunda sobrecarga se pasa como parámetro un objeto “Icon”. De esta forma podemos establecer como icono de la ventana una imagen Bitmap estándar o un archivo ICO (a diferencia del formulario base en .Net que únicamente admite archivos ICO). El icono siempre se mostrará en un tamaño de 32x32. Un ejemplo puede ser:

```
public Window1() : base(false)
{
      InitializeComponent();
      SetWindowTitle("Test of SwanCSharp", Brushes.White);
      SetIconWindowTitle(Properties.Resources.SwanCSharp_Solo_Logo_Grande);
}
```

Se puede comprobar que nuestra imagen de ejemplo se ha incluido en el archivo de recursos del proyecto, según lo explicado en el apartado “Imágenes e iconos en las ventanas SwanCSharp” incluido más arriba en este documento.

Con la línea “SetIconWindowTitle” incluida debajo de “InitializeComponent” la ventana que se mostrará al ejecutar la solución será la siguiente:

11.1.5.3 Botones de control de la ventana

Los botones de control de la ventana estándar WindowBase (cerrar, maximizar, minimizar, y ocultar) no se muestran si no se ejecutan los métodos expresamente, y esto nos permite plena libertad para definir los botones, ya que en los desarrollos estándar en .Net solo permite deshabilitar todos los botones a la vez (lo que nos impide establecer un icono en la barra de título), o desactivar de forma independiente los iconos maximizar y minimizar sin poder ocultar el botón de cerrar. Con la clase WindowBase vamos a poder desactivar los tres botones por separado y sin perder otros recursos, utilizando los métodos ShowCloseButton, ShowMaximizeButton, ShowMinimizeButton, and ShowNotifyButton.

```
public Window1() : base(false)
{
     InitializeComponent();

     SetIconWindowTitle(Properties.Resources.SwanCSharp_Solo_Logo_Grande1);

     SetWindowTitle("Test of SwanCSharp", Brushes.White);

     ShowCloseButton();
     ShowMaximizeButton();
     ShowMinimizeButton();
}
```

Según el ejemplo anterior habilitamos la ventana para que muestre los tres botones, pero si eliminamos cualquiera de los métodos, la ventana se mostrará sin el botón correspondiente, pudiendo ocultar los tres simultáneamente sin tener que perder la posibilidad de incluir el icono en la barra de título. Los botones se mostrarán de derecha a izquierda según el orden en el que se ejecuten sus métodos, y esto nos permite alterar el orden natural de esos botones. La ventana se mostraría así:

El botón de “Notificación” simplemente muestra un botón que ejecuta el comando “Hide()”, y es muy útil, por ejemplo, para aplicaciones de tipo “Tray application”.

11.1.5.4 Ventana “moveable”

Una característica desaparecida desde el Visual Basic 6.0 es la propiedad “Moveable” de las ventanas, pudiendo bloquear el arrastre de la ventana por toda la pantalla del Windows, quedando fija en la posición de arranque. En ninguna de las versiones de .Net existe la propiedad “Moveable” en los formularios (para bloquear el arrastre de una ventana hay que desarrollar clases personalizadas), pero en la clase “WindowBase” si existe un método que habilita/deshabilita el arrastre de la ventana: se llama IsMoveable.

```
public Window1() : base(false)
{
	InitializeComponent();

	IsMoveable(true);
}
```

En el ejemplo anterior la ventana se puede arrastrar y cambiar de posición en la pantalla de Windows.

```
public Window1() : base(false)
{
	InitializeComponent();

	IsMoveable(false);
}
```

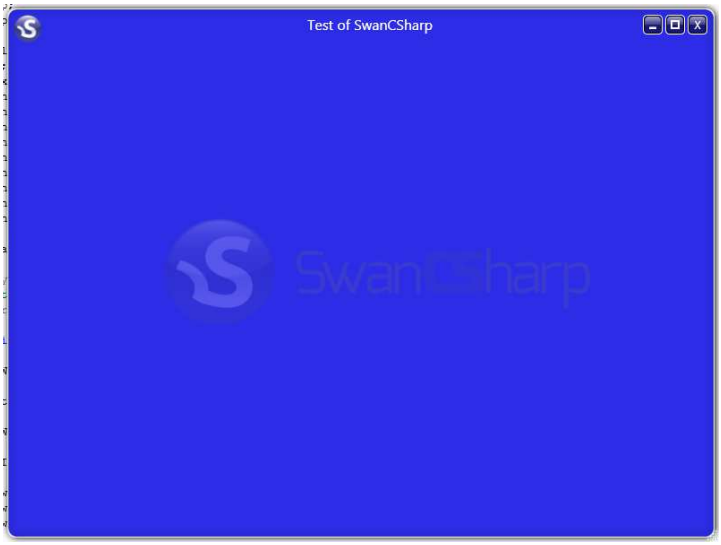
En el ejemplo anterior la ventana NO se puede arrastrar y cambiar de posición en la pantalla de Windows.

11.1.5.5 Marca de agua

En los formularios “SwanCSharp” se puede incluir una marca de agua de fondo de formulario con una imagen concreta. En la clase “WindowBase” existe el método “ShowWaterMark” que nos permite incluir la marca de agua; necesita ocho parámetros que son: objeto Bitmap de la imagen, el ancho de la imagen, el alto de la imagen, la posición inicial (utiliza el enumerado WaterMarkStartPosition existente en WindowBase), y los cuatro valores para establecer un margen (izquierda, arriba, derecha, abajo) y poder mover libremente la marca de agua por el formulario. Un ejemplo puede ser:

```
public Window1() : base(false)
{
     InitializeComponent();

     SetIconWindowTitle(Properties.Resources.SwanCSharp_Solo_Logo_Grande1);

     SetWindowTitle("Test of SwanCSharp", Brushes.White);

     ShowCloseButton();
     ShowMaximizeButton();
     ShowMinimizeButton();
}
```

En el ejemplo anterior se utiliza una imagen cargada como recurso en el proyecto C#, y se establece que la posición inicial es el centro del formulario, pasando los cuatro valores del margen con valor “cero” ya que no se desea mover dicha marca de agua de la posición central inicial. El formulario resultante es el siguiente:



11.1.6 Controles y objetos para las ventanas “SwanCSharp”

En este apartado se van a describir una serie de controles y objetos creados exclusivamente para las ventanas “SwanCSharp” y que podemos utilizar libremente en nuestros desarrollos.

Es necesario aclarar que todas las funciones definidas en este apartado devuelven un objeto de tipo control o un conjunto de controles (StackPanel). Si deseamos crear esos controles en nuestras ventanas “SwanCSharp” deberemos crear los objetos a nivel de variable de módulo para que podamos en cualquier momento acceder a sus propiedades y métodos. Por ejemplo:

```
public partial class Window1 : WindowBase
{
   private TextBox mtxtTextBox;

   public Window1() : base(false)
   {
        mtxtTextBox = new TextBox();
        mtxtTextBox = CreateWindowTextBox("txtTextBox", 150, 24, 20, 0, 0,
0, TextAlignment.Left, true);
        MainGrid.Children.Add(mtxtTextBox);

        if (mtxtTextBox.Text = "Test")
        {
        }
   }
}
```

En el ejemplo anterior declaramos el control “mtxtTextBox” a nivel de módulo, así podremos acceder a sus métodos y propiedades desde cualquier procedimiento de la ventana “Window1”.

En cambio si el objeto devuelto por las funciones de controles personalizadas no es un control concreto, sino un conjunto de controles (StackPanel), deberemos crear el conjunto de controles a nivel de módulo también, y deberemos acceder a sus elementos hijo para alcanzar sus métodos y propiedades. Por ejemplo:

```
public partial class Window1 : WindowBase
{
   private StackPanel mobjUser;

   public Window1() : base(false)
   {
        mobjUser = new StackPanel();
        mobjUser = CreateWindowTextBoxWithLabel("txtUser", 135, 24, 6, 60,
0, 0, TextAlignment.Left, true, "lblUser", "User", 123, Colors.White,
TextAlignment.Right);

        MainGrid.Children.Add(mobjUser);

        mobjUser.Children[1].Focus();
    }
}
```

En el ejemplo anterior declaramos el control StackPanel “mobjUser” a nivel de módulo, así podremos acceder a sus métodos y propiedades desde cualquier procedimiento de la ventana “Window1”. Posteriormente creamos en el StackPanel una caja de texto y una etiqueta, por lo tanto los elementos “hijo” de “mobjUser” son el elemento “0” (TextBlock) y el elemento “1” (TextBox). Queremos acceder al método “Focus()” del TextBox (segundo elemento), y para ello en el ejemplo utilizamos el método “Children” del objeto “StackPanel” principal.

Cuando los objetos utilizados son StackPanel posiblemente no tengamos acceso a todos métodos de cada objeto “hijo”, por ejemplo NO podemos acceder directamente a la propiedad “Text” de un objeto “TextBox” mediante el método “children” de “StackPanel”. Por

ejemplo si tenemos un StackPanel “stkUser” cuyo primer elemento “hijo” es un “TextBlock” y segundo elemento es un “TextBox”, si queremos limpiar la caja de texto podemos hacer:

```
TextBox lobjUser = (TextBox)stkUser.Children[1];
lobjUser.Text = "";
```

O también podemos recurrir a “SetValue”:

```
stkUser.Children[1].SetValue(TextBox.TextProperty, "");
```

A veces nos podemos encontrar algún caso particular en el que SetValue no se pueda utilizar. Por ejemplo si tenemos un StackPanel “stkPassword” cuyo primer elemento “hijo” es un “TextBlock” y segundo elemento es un “PasswordBox”, si queremos limpiar la caja de texto no podremos acceder a la propiedad “Password” desde “SetValue” por lo tanto únicamente podremos limpiarla mediante:

```
PasswordBox lobjPassword = (PasswordBox)stkPassword.Children[1];
lobjPassword.Password = "";
```

11.1.6.1 Botón de comando

En la clase “WindowBase” disponemos de un control “Button” personalizado con el estilo acorde a la ventana “SwanCSharp”. Existen dos funciones que nos devuelven objetos “Button” personalizados: “CreateWindowButton” y “CreateWindowButtonWithImage”. En la primera función solo se crean botones que muestran un texto, y se pasan un total de 9 parámetros: El texto a mostrar en el botón (content), el nombre que va a tener el control, el ancho del botón, el alto del botón, y los 4 siguientes parámetros son los márgenes a aplicar (izquierda, arriba, derecha, abajo) para colocar el botón en el lugar deseado dentro del formulario; en el último parámetro se pasa el objeto “RoutedEventHandler” que contiene el procedimiento donde se realizará el proceso al hacer clic sobre el botón. En la segunda función (CreateWindowButtonWithImage) se muestra un texto y una imagen en el botón, y tenemos los mismos parámetros que la primera, pero añadiendo otros tres parámetros, para definir la imagen que va a insertar y su ancho y alto. Dentro de esta segunda función existen dos sobrecargas, en la primera podemos pasar un objeto “Bitmap”, y en la segunda podemos pasar un objeto “Icon”.

A continuación se muestra un ejemplo utilizando la función “CreateWindowButton”, sin incluir una imagen en la visualización del botón.

```
public Window1() : base(false)
{
     InitializeComponent();

     SetWindowTitle("Test of SwanCSharp", Brushes.White);

     SetIconWindowTitle(Properties.Resources.SwanCSharp_Solo_Logo_Grande1);

     ShowCloseButton();
     ShowMaximizeButton();
     ShowMinimizeButton();

     IsMoveable(true);

     ShowWaterMark(Properties.Resources.SwanCSharp_Nuevo_Grande, 512, 190,
WaterMarkStartPosition.Center, 0, 0, 0, 0);
```

```
      Button lobjButton = CreateWindowButton("Execute", "cmdExecute", 130,
25, 215, 150, 0, 0, cmdExecute_Click);
       lobjButton.Click += new RoutedEventHandler(cmdExecute_Click);

       MainGrid.Children.Add(lobjButton);
}

private void cmdExecute_Click(object sender, RoutedEventArgs e)
{
    /*
    **** Insert our code here for Click event
    */
}
```

En el código se puede comprobar que llamamos a la función “CreateWindowButton” que nos devuelve un control “Button”. Previamente hemos creado la función “cmdExecute_Click” con el código a ejecutar cuando se pulse sobre el botón. En la última línea se incluye el objeto “Button” en la propiedad “MainGrid” de nuestro formulario. El resultado es el siguiente:

A continuación se muestra un ejemplo utilizando la función CreateWindowButtonWithImage, incluyendo un archivo de icono (ICO) en la visualización del botón (el archivo ICO se ha incluido en los recursos del proyecto C# de ejemplo).

```
public Window1() : base(false)
{
     InitializeComponent();

     SetWindowTitle("Test of SwanCSharp", Brushes.White);

     SetIconWindowTitle(Properties.Resources.SwanCSharp_Solo_Logo_Grande1);

     ShowCloseButton();
     ShowMaximizeButton();
     ShowMinimizeButton();

     IsMoveable(true);
```

```
     ShowWaterMark(Properties.Resources.SwanCSharp_Nuevo_Grande, 512, 190,
WaterMarkStartPosition.Center, 0, 0, 0, 0);

     Button lobjButton = CreateWindowButtonWithImage("Execute",
"cmdExecute", 130, 25, 215, 150, 0, 0,
Properties.Resources.SwanCSharp_Solo_Logo_Grande, 16, 16,
cmdExecute_Click);

     MainGrid.Children.Add(lobjButton);
}

private void cmdExecute_Click(object sender, RoutedEventArgs e)
{
    /*
    **** Insert our code here for Click event
    */
}
```

En el código se puede comprobar que llamamos a la función "CreateWindowButtonWithImage" que nos devuelve un control "Button". Previamente hemos creado la función "cmdExecute_Click" con el código a ejecutar cuando se pulse sobre el botón. En la última línea se incluye el objeto "Button" en la propiedad "MainGrid" de nuestro formulario. El resultado es el siguiente:

11.1.6.2 Check Box

En la clase "WindowBase" disponemos de un control "CheckBox" personalizado con el estilo acorde a la ventana "SwanCSharp". Existe una función que nos devuelve un objeto "CheckBox" personalizado: "CreateWindowCheckBox" que dispone de un total de 9 parámetros. El texto a mostrar en el control (content), el nombre que va a tener el control, el ancho, el alto, y los 4 siguientes parámetros son los márgenes a aplicar (izquierda, arriba, derecha, abajo) para colocar el control en el lugar deseado dentro del formulario; en el último parámetro se indica si se desea un efecto "sombra" para el formulario.

A continuación se muestra un ejemplo utilizando la función "CreateWindowCheckBox":

```
public Window1() : base(false)
{
     InitializeComponent();

     SetWindowTitle("Test of SwanCSharp", Brushes.White);

     SetIconWindowTitle(Properties.Resources.SwanCSharp_Solo_Logo_Grande1);

     ShowCloseButton();
     ShowMaximizeButton();
     ShowMinimizeButton();

     IsMoveable(true);

     ShowWaterMark(Properties.Resources.SwanCSharp_Nuevo_Grande, 512, 190,
WaterMarkStartPosition.Center, 0, 0, 0, 0);

     CheckBox chkTest = CreateWindowCheckBox("Check Test", "chkTest", 150,
24, 223, 160, 0, 0, true);
     MainGrid.Children.Add(chkTest);
}
```

El resultado es el siguiente:

11.1.6.3 Menú de Opciones

En la clase “WindowBase” disponemos de un control personalizado que nos permite incluir un menú de opciones tipo (Pop-Up) con todas las funcionalidades necesarias, permitiendo también incluir algunas características especiales, como por ejemplo un icono para el menú principal horizontal, además de permitir incluir los habituales iconos gráficos en las opciones de menú verticales en “Pop-Up”.
Para crear un menú en una ventana “SwanCSharp” vamos a utilizar la función “CreateWindowMenus” que nos devolverá un objeto “Menu” de .Net que agregaremos al MainGrid de nuestra ventana.

El control “Menu” de SwanCSharp permite crear una primera fila de opciones de menú horizontales que únicamente abren ventanas “Popup” con más opciones de menú verticales, permitiendo agregar recursivamente todas las opciones y submenús deseados.
También se permite la inclusión de iconos gráficos a la izquierda de cada opción de menú, permitiendo esos iconos también en las opciones de menú principal en horizontal. La función nos permite activar/desactivar mostrar los iconos en las opciones verticales y horizontales por separado.

Para crear un menú el primer paso es crear un objeto de tipo “MenuOptions” que existe en el espacio de nombres “SwanCSharp_Controls”; el objeto “MenuOptions” es una estructura de cinco variables: La variable “NameMainMenuControl” en la cual definidos el nombre interno para el control; la segunda variable es “NameToDisplayMainMenu” que es el nombre de la opción de menú que se mostrará en pantalla; la tercera variable es “IconToDisplay” el cual informamos de un icono gráfico mediante un objeto “Icon”; la cuarta variable es “SubmenuOptions” que es un nuevo objeto “MenuOptions” que podemos crear para incluir en el menú que estamos creando un segundo menú que dependerá de éste, mediante esta cuarta variable podemos crear los submenús que queramos recursivamente; la quinta variable es “MenuClickEvent” que es una variable en la que informamos un objeto “RoutedEventHandler”, que es la procedimiento que se ejecutará cuando se pulse en la opción de menú (evento clic).

Vamos a visualizar un menú de ejemplo, y para ello vamos a imaginar que queremos un menú de opciones con una opción en horizontal llamada “File” que al pulsar sobre ella nos va a mostrar una ventana “Popup” con una opción de submenú llamada “New”.
El primer paso es identificar en cuántas opciones vamos a tener que gestionar un procedimiento que se va a ejecutar al hacer clic. En nuestro ejemplo en la opción “File” no se va a gestionar un clic, ya que muestra el submenú “New”, por lo tanto no necesitamos un evento “RoutedEventHandler”. En cambio para la opción “New” si vamos a necesitar un procedimiento para incluir el código a ejecutar cuando se haga clic. El primer paso es crear el objeto “RouterEventHandler” para la opción “New”:

```
private void mnuNew_Click(object sender, RoutedEventArgs e)
{

}
```

Dentro del constructor de la ventana “Window1” vamos a crear el objeto “MenuOptions” que va a definir el menú. Primero creamos el menú principal vertical “File”:

```
public Window1() : base(false)
{
     InitializeComponent();

     SetWindowTitle("Test of SwanCSharp", Brushes.White);

     SetIconWindowTitle(Properties.Resources.SwanCSharp_Solo_Logo_Grande1);

     ShowCloseButton();
     ShowMaximizeButton();
     ShowMinimizeButton();

     IsMoveable(true);

     MenuOption[] lobjMenuOptions = new MenuOption[1];
     // First option in Main Menu
```

```
        lobjMenuOptions[0].NameMainMenuControl = "File";
        lobjMenuOptions[0].NameToDisplayMainMenu = "File";
        lobjMenuOptions[0].IconToDisplay = Properties.Resources.Floppy_Drive;
}
```

En el código fuente anterior se puede comprobar que creamos un array de “MenuOption” con un único elemento, ya que en el menú horizontal solo tenemos la opción “File”. Para el icono gráfico utilizamos Floppy_Drive que previamente lo hemos cargado como recurso del proyecto de tipo “Icon”. El siguiente paso es crear un segundo objeto “MenuOption” diferente para el submenú “New”:

```
MenuOption[] lobjSubMenuFileOption = new MenuOption[1];
lobjSubMenuFileOption[0].NameMainMenuControl = "New";
lobjSubMenuFileOption[0].NameToDisplayMainMenu = "New";
lobjSubMenuFileOption[0].IconToDisplay = Properties.Resources.newfolder;
lobjSubMenuFileOption[0].MenuClickEvent += mnuNew_Click;
```

En el código fuente anterior se ve como relacionamos el procedimiento “mnuNew_Click” con el evento clic de esta opción de menú (New). Ahora deseamos que la opción “New” sea un submenú “Popup” que se abra al pulsar sobre “File”. Por lo tanto el objeto “lobjSubmenuFileOption” pasa a ser un objeto de “lobjMenuOptions”.

```
lobjMenuOptions[0].SubMenuOptions = lobjSubMenuFileOption;
```

Con esto ya hemos construido el objeto “MenuOptions” para crear el menú, el siguiente paso es llamar al método que va a construir el objeto necesitado y después lo añadimos al “MainGrid” de nuestra ventana “SwanCSharp”:

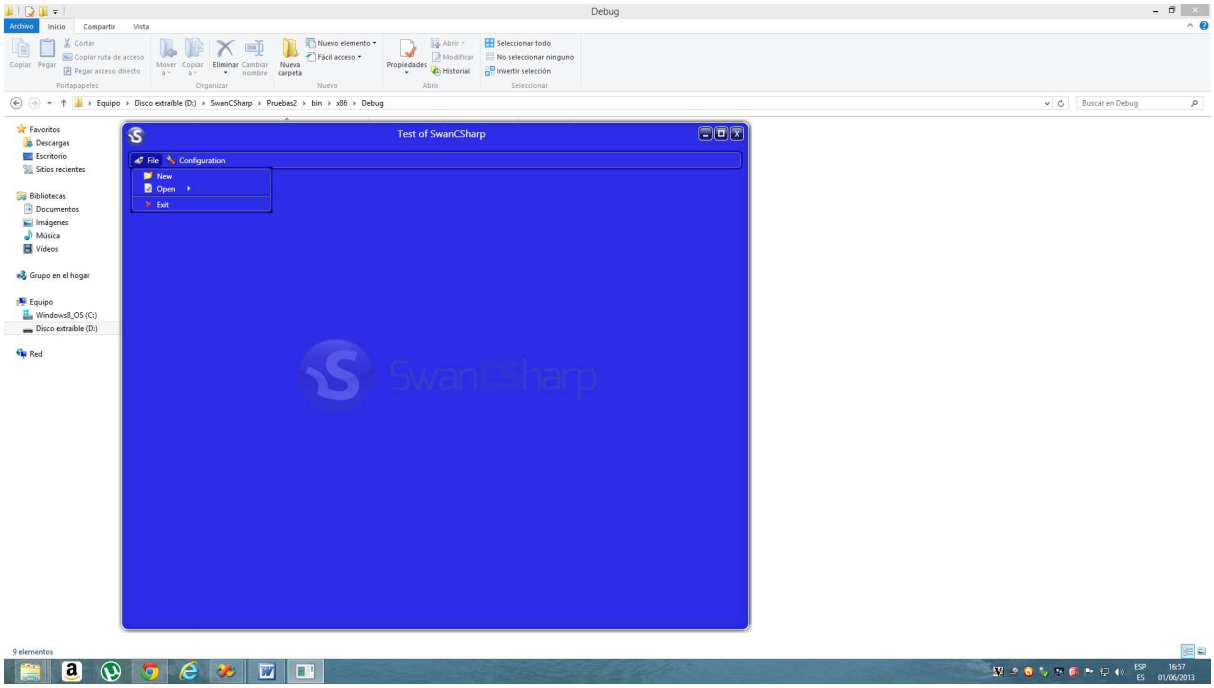
```
Menu lobjMenu = CreateWindowMenus(lobjMenuOptions, true, true);
MainGrid.Children.Add(lobjMenu);
```

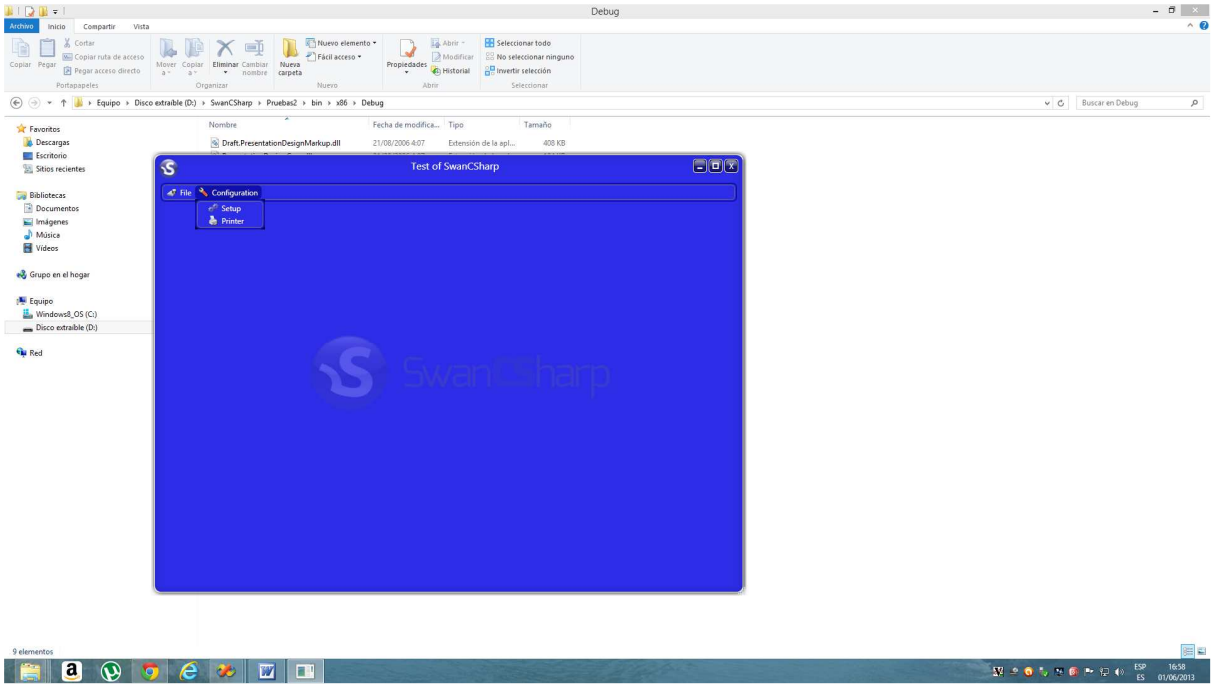
En el primer parámetro de “CreateWindowMenus” pasamos el objeto “MenuOption” creado, en el segundo parámetro le indicamos si queremos mostrar los iconos en el menú principal horizontal, y en el tercer parámetro pasamos si deseamos mostrar los iconos en todas las opciones verticales de menú. El código completo de la ventana quedaría así:

```
using SwanCSharp;
using SwanCSharp_Controls;
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Markup;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace Test2
{
    public partial class Window1 : WindowBase
    {
```

```
        public Window1() : base(false)
        {
            InitializeComponent();

            SetWindowTitle("Test of SwanCSharp", Brushes.White);
     SetIconWindowTitle(Properties.Resources.SwanCSharp_Solo_Logo_Grande1);

            ShowCloseButton();
            ShowMaximizeButton();
            ShowMinimizeButton();

            ShowWaterMark(Properties.Resources.SwanCSharp_Nuevo_Grande,
512, 190, WaterMarkStartPosition.Center, 0, 0, 0, 0);

            IsMoveable(true);

            MenuOption[] lobjMenuOptions = new MenuOption[1];
            // First option in Main Menu
            lobjMenuOptions[0].NameMainMenuControl = "File";
            lobjMenuOptions[0].NameToDisplayMainMenu = "File";
            lobjMenuOptions[0].IconToDisplay =
Properties.Resources.Floppy_Drive;
            // Suboptions in First Submenu File
            MenuOption[] lobjSubMenuFileOption = new MenuOption[1];
            lobjSubMenuFileOption[0].NameMainMenuControl = "New";
            lobjSubMenuFileOption[0].NameToDisplayMainMenu = "New";
            lobjSubMenuFileOption[0].IconToDisplay =
Properties.Resources.newfolder;
            lobjSubMenuFileOption[0].MenuClickEvent += mnuNew_Click;
            lobjMenuOptions[0].SubMenuOptions = lobjSubMenuFileOption;

            Menu lobjMenu = CreateWindowMenus(lobjMenuOptions, true, true);

            MainGrid.Children.Add(lobjMenu);
        }

        private void mnuNew_Click(object sender, RoutedEventArgs e)
        {
            /*
             * Source code to execute New option event click.
             */
        }
    }
}
```

Crear un menú “SwanCSharp” se basa en crear objetos “MenuOptions” que se van agregando recursivamente a esa estructura principal para terminar por llamar a la función “CreateWindowMenus” que nos devolverá el objeto “Menu” construido y que agregaremos al “MainGrid” de nuestra ventana. A continuación vamos a mostrar un ejemplo de menú más completo:






```
using SwanCSharp;
using SwanCSharp_Controls;
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Markup;
```

```
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace Test2
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>

    public partial class Window1 : WindowBase
    {
        public Window1() : base(false)
        {
            InitializeComponent();

            SetWindowTitle("Test of SwanCSharp", Brushes.White);


SetIconWindowTitle(Properties.Resources.SwanCSharp_Solo_Logo_Grande1);

            ShowCloseButton();
            ShowMaximizeButton();
            ShowMinimizeButton();

            ShowWaterMark(Properties.Resources.SwanCSharp_Nuevo_Grande,
512, 190, WaterMarkStartPosition.Center, 0, 0, 0, 0);

            IsMoveable(true);

            MenuOption[] lobjMenuOptions = new MenuOption[2];
            // First option in Main Menu
            lobjMenuOptions[0].NameMainMenuControl = "File";
            lobjMenuOptions[0].NameToDisplayMainMenu = "File";
            lobjMenuOptions[0].IconToDisplay =
Properties.Resources.Floppy_Drive;
            // Suboptions in First Submenu File
            MenuOption[] lobjSubMenuFileOption = new MenuOption[4];
            lobjSubMenuFileOption[0].NameMainMenuControl = "New";
            lobjSubMenuFileOption[0].NameToDisplayMainMenu = "New";
            lobjSubMenuFileOption[0].IconToDisplay =
Properties.Resources.newfolder;
            lobjSubMenuFileOption[0].MenuClickEvent += mnuNew_Click;
            lobjSubMenuFileOption[1].NameMainMenuControl = "Open";
            lobjSubMenuFileOption[1].NameToDisplayMainMenu = "Open";
            lobjSubMenuFileOption[1].MenuClickEvent += mnuOpen_Click;
            lobjSubMenuFileOption[1].IconToDisplay =
Properties.Resources.propertiesORoptions;
            // Third level suboptions for the submenu option "Open"
            MenuOption[] lobjSubMenuNewOption = new MenuOption[2];
            lobjSubMenuNewOption[0].NameMainMenuControl = "File";
            lobjSubMenuNewOption[0].NameToDisplayMainMenu = "File";
            lobjSubMenuNewOption[0].IconToDisplay =
Properties.Resources.Floppy_Drive;
            lobjSubMenuNewOption[1].NameMainMenuControl = "Folder";
            lobjSubMenuNewOption[1].NameToDisplayMainMenu = "Folder";
            lobjSubMenuNewOption[1].IconToDisplay =
Properties.Resources.newfolder;
            lobjSubMenuFileOption[1].SubMenuOptions = lobjSubMenuNewOption;
```

```
            lobjMenuOptions[0].SubMenuOptions = lobjSubMenuFileOption;

            lobjSubMenuFileOption[2].NameMainMenuControl = "-";
            lobjSubMenuFileOption[2].NameToDisplayMainMenu = "-";
            lobjSubMenuFileOption[3].NameMainMenuControl = "Exit";
            lobjSubMenuFileOption[3].NameToDisplayMainMenu = "Exit";
            lobjSubMenuFileOption[3].MenuClickEvent += mnuExit_Click;
            lobjSubMenuFileOption[3].IconToDisplay =
Properties.Resources.delete_16x;

            // Second option in Main Menu
            lobjMenuOptions[1].NameMainMenuControl = "Configuration";
            lobjMenuOptions[1].NameToDisplayMainMenu = "Configuration";
            lobjMenuOptions[1].IconToDisplay = Properties.Resources.repair;
            // Suboptions in Second Submenu File
            MenuOption[] lobjSubMenuSetupOption = new MenuOption[2];
            lobjSubMenuSetupOption[0].NameMainMenuControl = "Setup";
            lobjSubMenuSetupOption[0].NameToDisplayMainMenu = "Setup";
            lobjSubMenuSetupOption[0].IconToDisplay =
Properties.Resources.services;
            lobjSubMenuSetupOption[1].NameMainMenuControl = "Printer";
            lobjSubMenuSetupOption[1].NameToDisplayMainMenu = "Printer";
            lobjSubMenuSetupOption[1].IconToDisplay =
Properties.Resources.printer;
            lobjMenuOptions[1].SubMenuOptions = lobjSubMenuSetupOption;

            Menu lobjMenu = CreateWindowMenus(lobjMenuOptions, true, true);

            MainGrid.Children.Add(lobjMenu);
        }

        private void mnuNew_Click(object sender, RoutedEventArgs e)
        {
            /*
             * Source code to execute New option event click.
             */
        }

        private void mnuOpen_Click(object sender, RoutedEventArgs e)
        {
        }

        private void mnuExit_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
        }
    }
}
```

11.1.6.1 ListBox

En la clase “WindowBase” disponemos de un control “ListBox” personalizado con el estilo acorde a la ventana “SwanCSharp”. Existen dos funciones que nos devuelven objetos “ListBox” y “StackPanel” personalizados (según función elegida): “CreateWindowListBox” y “CreateWindowListBoxWithLabel”. Ambas funciones permiten seleccionar si se desea un efecto sombra para el control, el tamaño, etcétera.

La primera función (CreateWindowListBox) únicamente crea el control lista, y se pasan un total de 8 parámetros: El nombre que va a tener el control, el ancho del control, el alto del control, y los 4 siguientes parámetros son los márgenes a aplicar (izquierda, arriba, derecha, abajo) para colocar el control en el lugar deseado dentro del formulario; en el último parámetro se indica si se desea mostrar un efecto sombra.

En la segunda función (CreateWindowListBoxWithLabel) se crea un objeto StackPanel que integra la etiqueta de texto a mostrar y el control lista, todo en un único objeto. La función consta de trece parámetros: los ocho primeros parámetros para definir el "ListBox" y los cinco últimos parámetros para definir el "LabelBox". Los ocho primeros parámetros son exactamente iguales que en la función "CreateWindowLabelBox"; los cinco últimos parámetros son: El nombre que va a tener el control, el texto a mostrar en la etiqueta, el alto del control, el color del texto de la etiqueta y la alineación del texto dentro de la etiqueta.

A continuación se muestra el código fuente de un ejemplo de un "ListBox:

```
public Window1() : base(false)
{
     InitializeComponent();

     SetWindowTitle("Test of SwanCSharp", Brushes.White);

     SetIconWindowTitle(Properties.Resources.SwanCSharp_Solo_Logo_Grande1);

     ShowCloseButton();
     ShowMaximizeButton();
     ShowMinimizeButton();

     ShowWaterMark(Properties.Resources.SwanCSharp_Nuevo_Grande, 512, 190,
WaterMarkStartPosition.Center, 0, 0, 0, 0);

     IsMoveable(true);

     StackPanel stkList = CreateWindowListBoxWithLabel("lstList", 200, 150,
20, 80, 0, 0, true, "lblList", "List", 24, Colors.Black,
TextAlignment.Left);

     ListBox lstList = (ListBox)stkList.Children[1];

     MainGrid.Children.Add(stkList)
}
```

11.1.6.2 TextBox

En la clase "WindowBase" disponemos de un control "TextBox" personalizado con el estilo acorde a la ventana "SwanCSharp". Existen dos funciones que nos devuelven objetos "TextBox" y "StackPanel" personalizados (según función elegida): "CreateWindowTextBox" y "CreateWindowTextBoxWithLabel". Ambas funciones permiten seleccionar si se desea un efecto sombra para el control, el tamaño, la alineación del texto, etcétera.

La primera función (CreateWindowTextBox) únicamente crea la caja de texto, y se pasan un total de 9 parámetros: El nombre que va a tener el control, el ancho del control, el alto del control, y los 4 siguientes parámetros son los márgenes a aplicar (izquierda, arriba, derecha, abajo) para colocar el control en el lugar deseado dentro del formulario; en el octavo

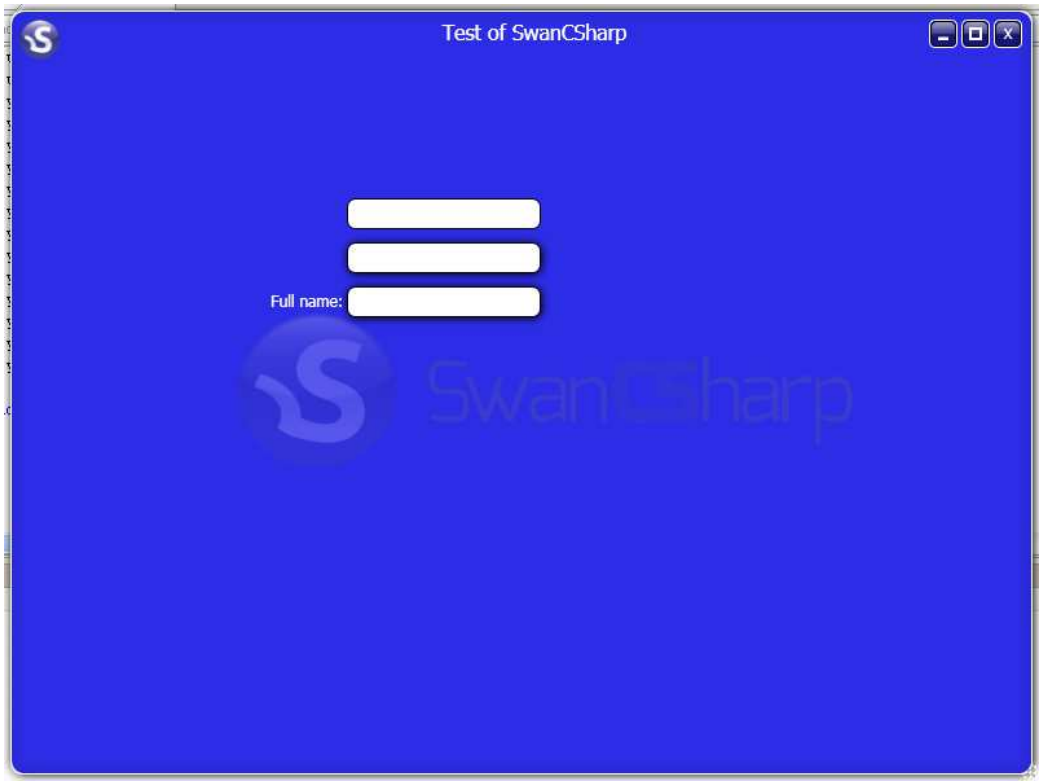
parámetro se pasa la alineación del texto elegida; en el último parámetro se indica si se desea mostrar un efecto sombra.

En la segunda función (CreateWindowTextBoxWithLabel) se crea un objeto StackPanel que integra la etiqueta de texto a mostrar y la caja de texto, todo en un único objeto. La función consta de catorce parámetros: los nueve primeros parámetros para definir el "TextBox" y los cinco últimos parámetros para definir el "LabelBox". Los nueve primeros parámetros son exactamente iguales que en la función "CreateWindowTextBox"; los cinco últimos parámetros son: El nombre que va a tener el control, el texto a mostrar en la etiqueta, el ancho del control, el color del texto de la etiqueta y la alineación del texto dentro de la etiqueta.

A continuación se muestra el código fuente de un ejemplo de tres cajas de texto en varias combinaciones (sin etiqueta y sombra, con sombra sin etiqueta, y con sombra y etiqueta):

```
public Window1() : base(false)
{
     InitializeComponent();

     SetWindowTitle("Test of SwanCSharp", Brushes.White);

     SetIconWindowTitle(Properties.Resources.SwanCSharp_Solo_Logo_Grande1);

     ShowCloseButton();
     ShowMaximizeButton();
     ShowMinimizeButton();

     ShowWaterMark(Properties.Resources.SwanCSharp_Nuevo_Grande, 512, 190,
WaterMarkStartPosition.Center, 0, 0, 0, 0);

     IsMoveable(true);


     TextBox lobjTextBox = CreateWindowTextBox("txtTextBox", 150, 24, 265,
150, 0, 0, TextAlignment.Left, false);
     MainGrid.Children.Add(lobjTextBox);

     TextBox lobjTextBoxShadow =
CreateWindowTextBox("txtTextBoxWithShadow", 150, 24, 265, 184, 0, 0,
TextAlignment.Right, true);
     MainGrid.Children.Add(lobjTextBoxShadow);

     StackPanel lobjTextBoxWithLabel =
CreateWindowTextBoxWithLabel("txtTextBoxWithLabel", 150, 24, 185, 218, 0,
0, TextAlignment.Right, true, "lblTextBox","Full name:", 80, Colors.White,
TextAlignment.Right);
     MainGrid.Children.Add(lobjTextBoxWithLabel);
}
```

Según el código fuente anterior, al ejecutar la aplicación, se mostraría en pantalla el siguiente formulario "SwanCSharp":



11.1.6.3 PasswordBox

En la clase “WindowBase” disponemos de un control “PasswordBox” personalizado con el estilo acorde a la ventana “SwanCSharp”. Existen dos funciones que nos devuelven objetos “PasswordBox” y “StackPanel” personalizados (según función elegida): “CreateWindowPasswordBox” y “CreateWindowPasswordBoxWithLabel”. Ambas funciones realizan exactamente el mismo proceso que las funciones CreateWindowTextBox” y “CreateWindowTextBoxWithLabel” con la diferencia de que en vez de generar un “TextBox” estándar, generan una caja de texto especial para introducir contraseñas.

11.1.6.1 LabelData

En la clase “WindowBase” disponemos de un control “LabelData” personalizado con el estilo acorde a la ventana “SwanCSharp”. La función de creación del control devuelve un objeto “StackPanel”.

La función “CreateWindowLabelData” está pensada para crear un control que muestre texto divido en dos partes: Etiqueta y dato. La idea es mostrar datos pudiendo dar un color diferente a la etiqueta y al dato. Se pasan un total de quince parámetros: El nombre que va a tener el control, el alto del control, el tamaño de la fuente de letra, los 4 siguientes parámetros son los márgenes a aplicar (izquierda, arriba, derecha, abajo) para colocar el control en el lugar deseado dentro del formulario, el texto a mostrar en la parte de la etiqueta, el ancho de la etiqueta, la alineación del texto en la etiqueta, el color de la etiqueta, el texto a mostrar en la parte del dato, el ancho del dato, la alineación del texto del dato, y el color del texto del dato.

A continuación se muestra el código fuente de un ejemplo de dos controles “LabelData” sobre un formulario “SwanCSharp”:

```
public Window1() : base(false)
{
     InitializeComponent();

     SetWindowTitle("Test of SwanCSharp", Brushes.White);

     SetIconWindowTitle(Properties.Resources.SwanCSharp_Solo_Logo_Grande1);

     ShowCloseButton();
     ShowMaximizeButton();
     ShowMinimizeButton();

     ShowWaterMark(Properties.Resources.SwanCSharp_Nuevo_Grande, 512, 190,
WaterMarkStartPosition.Center, 0, 0, 0, 0);

     IsMoveable(true);


     StackPanel lobjStack = CreateWindowLabelData("lobjStack", 24, 14, 100,
100, 0, 0, "Label:", 80, TextAlignment.Left, Brushes.White, "150067", 140,
TextAlignment.Left, Brushes.Red);
     MainGrid.Children.Add(lobjStack);

     StackPanel lobjStack2 = CreateWindowLabelData("lobjStack2", 24, 14,
100, 130, 0, 0, "Label 2:", 80, TextAlignment.Left, Brushes.White,
"678921", 140, TextAlignment.Left, Brushes.Red);
     MainGrid.Children.Add(lobjStack2);}
```

Según el código fuente anterior, al ejecutar la aplicación, se mostraría en pantalla el siguiente formulario “SwanCSharp”:

Si se desea cambiar algún dato durante la ejecución de nuestro desarrollo podemos hacerlo de la siguiente forma:

```
TextBlock lobjText = (TextBlock)lobjStack.Children[1];
lobjText.Text = "111222";
```

11.1.6.2 GroupBox

En la clase “WindowBase” disponemos de un control “GroupBox” personalizado con el estilo acorde a la ventana “SwanCSharp”. Existe una función que nos devuelve un objeto “GroupBox” personalizado: “CreateWindowGroupBox”. La función dispone de dos sobrecargas.

La primera sobrecarga dispone de tres parámetros: El nombre que va a tener el control, el texto del título del control, y un array del objeto “UIElement”. Esta sobrecarga creará un “GroupBox” que abarcará toda la ventana, evitando la barra de título y la zona reservada para el menú principal de opciones.

La segunda sobrecarga dispone de nueve parámetros: El nombre que va a tener el control, el texto del título del control, el ancho del control, el alto del control, los siguientes cuatro parámetros se corresponden al “Margin”, y un array del objeto “UIElement”. Esta segunda sobrecarga nos permite definir completamente del “GroupBox” el tamaño y la ubicación dentro del formulario.

El control “GroupBox” se utiliza para contener otros controles secundarios, y para definir los controles secundarios contenidos en el “GroupBox” se utiliza el objeto array “UIElement”. Por ejemplo, si queremos colocar tres cajas de texto dentro del GroupBox, antes de crear el GroupBox deberemos crear un array de tres objeto “UIElement”. Por ejemplo:

```
UIElement[] lobjUIElements = new UIElement[3];

TextBox lobjTextBox = CreateWindowTextBox("txtTextBox", 150, 24, 20, 0, 0,
0, TextAlignment.Left, false);
lobjUIElements[0] = lobjTextBox;

TextBox lobjTextBoxShadow = CreateWindowTextBox("txtTextBoxWithShadow",
150, 24, 20, 34, 0, 0, TextAlignment.Right, true);
lobjUIElements[1] = lobjTextBoxShadow;

StackPanel lobjTextBoxWithLabel =
CreateWindowTextBoxWithLabel("txtTextBoxWithLabel", 150, 24, 0, 68, 0, 0,
TextAlignment.Right, true, "lblTextBox", "Full name:", 80, Colors.White,
TextAlignment.Right);
lobjUIElements[2] = lobjTextBoxWithLabel;
```

Una vez creado el objeto array “UIElement” con todos los controles que tendrá el “GroupBox” en su interior, creamos el control “GroupBox” y lo añadimos a la propiedad “MainGrid” de la ventana:

```
GroupBox lobjGroupBox = new GroupBox();
lobjGroupBox = CreateWindowGroupBox("grpGroup", " Management ", 300, 200,
16, 85, 0, 0, lobjUIElements);

MainGrid.Children.Add(lobjGroupBox);
```

A continuación se muestra el código fuente completo del ejemplo anterior que muestra un "GroupBox" con tres cajas de texto en una posición del formulario concreta:

```
public Window1() : base(false)
{
    InitializeComponent();

    SetWindowTitle("Test of SwanCSharp", Brushes.White);

    SetIconWindowTitle(Properties.Resources.SwanCSharp_Solo_Logo_Grande1);

    ShowCloseButton();
    ShowMaximizeButton();
    ShowMinimizeButton();

    ShowWaterMark(Properties.Resources.SwanCSharp_Nuevo_Grande, 512, 190,
WaterMarkStartPosition.Center, 0, 0, 0, 0);

    IsMoveable(true);

    UIElement[] lobjUIElements = new UIElement[3];

    TextBox lobjTextBox = CreateWindowTextBox("txtTextBox", 150, 24, 20, 0,
0, 0, TextAlignment.Left, false);
    lobjUIElements[0] = lobjTextBox;

    TextBox lobjTextBoxShadow = CreateWindowTextBox("txtTextBoxWithShadow",
150, 24, 20, 34, 0, 0, TextAlignment.Right, true);
     lobjUIElements[1] = lobjTextBoxShadow;

     StackPanel lobjTextBoxWithLabel =
CreateWindowTextBoxWithLabel("txtTextBoxWithLabel", 150, 24, 0, 68, 0, 0,
TextAlignment.Right, true, "lblTextBox", "Full name:", 80, Colors.White,
TextAlignment.Right);
     lobjUIElements[2] = lobjTextBoxWithLabel;

     GroupBox lobjGroupBox = new GroupBox();
     lobjGroupBox = CreateWindowGroupBox("grpGroup", " Management ", 300,
200, 16, 85, 0, 0, lobjUIElements);

     MainGrid.Children.Add(lobjGroupBox);
}
```

Según el código fuente anterior, al ejecutar la aplicación, se mostraría en pantalla el siguiente formulario "SwanCSharp":

11.1.6.1 GroupLabelData

En la clase “WindowBase” disponemos de un control “GroupLabelData” personalizado con el estilo acorde a la ventana “SwanCSharp”. Existe una función que nos devuelve un objeto “GroupBox” personalizado: “CreateWindowGroupLabelData”.

La función "CreateWindowGroupLabelData" dispone de trece parámetros: El nombre que va a tener el control, el texto del título del control, el ancho del control, el alto del control, el tamaño de la fuente de letra, los siguientes cuatro parámetros se corresponden al “Margin”, el alto del conjunto “LabelData”, el ancho de la parte “Label”, el ancho de la parte “Data”, y el objeto array “LabelData” que contiene la información de cada línea de etiqueta de datos.

El primer paso es crear el array “LabelData”. En nuestro caso vamos a crear tres líneas de etiquetas con su correspondiente dato cada etiqueta. Por ejemplo:

```
LabelData[] lobjLabelData = new LabelData[3];

lobjLabelData[0].LabelText = "Label One:";
lobjLabelData[0].ColorLabelText = Brushes.White;
lobjLabelData[0].LabelAlignment = TextAlignment.Left;
lobjLabelData[0].DataText = "123487";
lobjLabelData[0].ColorDataText = Brushes.Red;
lobjLabelData[0].DataAlignment = TextAlignment.Left;

lobjLabelData[1].LabelText = "Label Two:";
lobjLabelData[1].ColorLabelText = Brushes.White;
lobjLabelData[1].LabelAlignment = TextAlignment.Left;
lobjLabelData[1].DataText = "765832";
```

```
lobjLabelData[1].ColorDataText = Brushes.Red;
lobjLabelData[1].DataAlignment = TextAlignment.Left;

lobjLabelData[2].LabelText = "Label Three:";
lobjLabelData[2].ColorLabelText = Brushes.White;
lobjLabelData[2].LabelAlignment = TextAlignment.Left;
lobjLabelData[2].DataText = "387496";
lobjLabelData[2].ColorDataText = Brushes.Red;
lobjLabelData[2].DataAlignment = TextAlignment.Left;
```

Una vez creado el objeto array “LabelData” con la información de cada línea, creamos el control “GroupLabelData” (que será un objeto GroupBox construido) y lo añadimos a la propiedad “MainGrid” de la ventana:

```
GroupBox lobjGroup = CreateWindowGroupLabelData("lobjGroup", "Label and
Data", 175, 140, 14, 100, 80, 0, 0, 24, 120, 100, lobjLabelData);

MainGrid.Children.Add(lobjGroup);
```

A continuación se muestra el código fuente completo del ejemplo anterior que muestra un “GroupLabelData” con líneas de texto:

```
public Window1() : base(false)
{
    InitializeComponent();

    SetWindowTitle("Test of SwanCSharp", Brushes.White);

    SetIconWindowTitle(Properties.Resources.SwanCSharp_Solo_Logo_Grande1);

    ShowCloseButton();
    ShowMaximizeButton();
    ShowMinimizeButton();

    ShowWaterMark(Properties.Resources.SwanCSharp_Nuevo_Grande, 512, 190,
WaterMarkStartPosition.Center, 0, 0, 0, 0);

    IsMoveable(true);

    LabelData[] lobjLabelData = new LabelData[3];

    lobjLabelData[0].LabelText = "Label One:";
    lobjLabelData[0].ColorLabelText = Brushes.White;
    lobjLabelData[0].LabelAlignment = TextAlignment.Left;
    lobjLabelData[0].DataText = "123487";
    lobjLabelData[0].ColorDataText = Brushes.Red;
    lobjLabelData[0].DataAlignment = TextAlignment.Left;

    lobjLabelData[1].LabelText = "Label Two:";
    lobjLabelData[1].ColorLabelText = Brushes.White;
    lobjLabelData[1].LabelAlignment = TextAlignment.Left;
    lobjLabelData[1].DataText = "765832";
    lobjLabelData[1].ColorDataText = Brushes.Red;
    lobjLabelData[1].DataAlignment = TextAlignment.Left;

    lobjLabelData[2].LabelText = "Label Three:";
    lobjLabelData[2].ColorLabelText = Brushes.White;
    lobjLabelData[2].LabelAlignment = TextAlignment.Left;
```

```
    lobjLabelData[2].DataText = "387496";
    lobjLabelData[2].ColorDataText = Brushes.Red;
    lobjLabelData[2].DataAlignment = TextAlignment.Left;

    GroupBox lobjGroup = CreateWindowGroupLabelData("lobjGroup", "Label and
Data", 175, 140, 14, 100, 80, 0, 0, 24, 120, 100, lobjLabelData);

    MainGrid.Children.Add(lobjGroup);
}
```

Según el código fuente anterior, al ejecutar la aplicación, se mostraría en pantalla el siguiente formulario “SwanCSharp”:

11.1.6.2 ComboBox

En la clase “WindowBase” disponemos de un control “ComboBox” personalizado con el estilo acorde a la ventana “SwanCSharp”. Existen dos funciones que nos devuelven objetos “ComboBox” y “StackPanel” personalizados (según función elegida): “CreateWindowComboBox” y “CreateWindowComboBoxWithLabel”. Ambas funciones permiten seleccionar si se desea un efecto sombra para el control, el tamaño, la alineación del texto, etcétera.

La primera función (CreateWindowComboBox) únicamente crea el control, y se pasan un total de 8 parámetros: El nombre que va a tener el control, el ancho del control, el alto del control, y los 4 siguientes parámetros son los márgenes a aplicar (izquierda, arriba, derecha, abajo) para colocar el control en el lugar deseado dentro del formulario; el último parámetro se indica si se desea mostrar un efecto sombra.

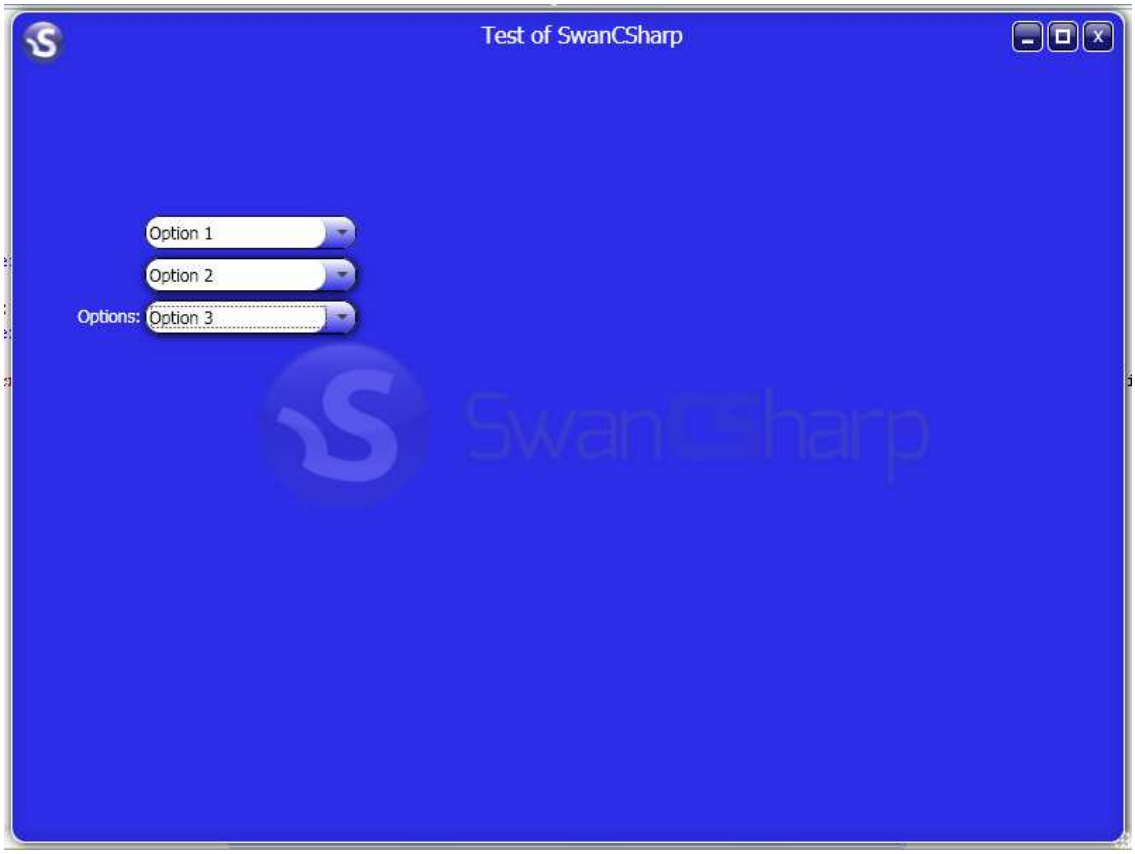
En la segunda función (CreateWindowComboBoxWithLabel) se crea un objeto StackPanel que integra la etiqueta de texto a mostrar y el control, todo en un único objeto. La función consta de quince parámetros: los ocho primeros parámetros para definir el “ComboBox”, los dos siguientes para cargar los datos en el control, y los cinco últimos parámetros para definir el “LabelBox”. Los ocho primeros parámetros son exactamente iguales que en la función

"CreateWindowComboBox"; los dos siguientes parámetros son de tipo "string[]" para pasar el dato y el índice de cada fila a cargar (si no se desean cargar datos, se pasan ambos parámetros como "**null**"), los cinco últimos parámetros son: El nombre que va a tener el control, el texto a mostrar en la etiqueta, el ancho del control, el color del texto de la etiqueta y la alineación del texto dentro de la etiqueta.

A continuación se muestra el código fuente de un ejemplo de tres ComboBox en varias combinaciones (sin etiqueta y sombra, con sombra sin etiqueta, y con sombra y etiqueta):

```
public Window1() : base(false)
{
     InitializeComponent();

     SetWindowTitle("Test of SwanCSharp", Brushes.White);

     SetIconWindowTitle(Properties.Resources.SwanCSharp_Solo_Logo_Grande1);

     ShowCloseButton();
     ShowMaximizeButton();
     ShowMinimizeButton();

     ShowWaterMark(Properties.Resources.SwanCSharp_Nuevo_Grande, 512, 190,
WaterMarkStartPosition.Center, 0, 0, 0, 0);

     IsMoveable(true);


     string[] lstrData = new string[3];
     string[] lstrIndex = new string[3];

     lstrData[0] = "Option 1";
     lstrIndex[0] = "1";

     lstrData[1] = "Option 2";
     lstrIndex[1] = "2";

     lstrData[2] = "Option 3";
     lstrIndex[2] = "3";

     ComboBox lobjComboBox = CreateWindowComboBox("cmbComboBox", 150, 24,
100, 150, 0, 0, false);
     SW_Miscellaneous.LoadDataInComboBox(lstrData, lstrIndex, ref
lobjComboBox);

     ComboBox lobjComboBox2 = CreateWindowComboBox("cmbComboBox2", 150, 24,
100, 180, 0, 0, true);
     SW_Miscellaneous.LoadDataInComboBox(lstrData, lstrIndex, ref
lobjComboBox2);

     StackPanel lobjComboBox3 =
CreateWindowComboBoxWithLabel("cmbComboBox3", 150, 24, 20, 210, 0, 0, true,
lstrData, lstrIndex,
"lblCombo","Options:",80,Colors.White,TextAlignment.Right);

     MainGrid.Children.Add(lobjComboBox);
     MainGrid.Children.Add(lobjComboBox2);
     MainGrid.Children.Add(lobjComboBox3);
}
```

Según el código fuente anterior, al ejecutar la aplicación, se mostraría en pantalla el siguiente formulario “SwanCSharp”:



11.1.6.3 DataGrid

En la clase “WindowBase” disponemos de un control “DataGrid” personalizado con el estilo acorde a la ventana “SwanCSharp”. Existe una función que nos devuelve un objeto “ListView” personalizado: “CreateWindowDataGrid”. La función dispone de nueve parámetros: El nombre que va a tener el control, el objeto “DataTable” que contiene las columnas a mostrar y los datos, el ancho del control, el alto del control, los siguientes cuatro parámetros se corresponden al “Margin”, y el último parámetro se indica si se desea un efecto de sombra al control. Un ejemplo de llamada a un control “DataGrid” puede ser:

```
DataConnectionAccess lobjConnection = new
DataConnectionAccess("Database.mdb", "", "", DatabaseManager.Access);
DataTable ldatData = lobjConnection.SQLSelectExecute("SELECT * FROM
Staff");
lobjConnection.CloseConnection();

ListView lobjListView = CreateWindowDataGrid("datListView", ldatData, 600,
300, 100, 150, 0, 0, true);

MainGrid.Children.Add(lobjListView);
```

A continuación se muestra el código fuente completo del ejemplo anterior que muestra un “DataGrid” con los datos existentes en una base de datos “Microsoft Access”:

```
public Window1() : base(false)
{
    InitializeComponent();

    SetWindowTitle("Test of SwanCSharp", Brushes.White);

    SetIconWindowTitle(Properties.Resources.SwanCSharp_Solo_Logo_Grande1);

    ShowCloseButton();
    ShowMaximizeButton();
    ShowMinimizeButton();

    ShowWaterMark(Properties.Resources.SwanCSharp_Nuevo_Grande, 512, 190,
WaterMarkStartPosition.Center, 0, 0, 0, 0);

    IsMoveable(true);

    DataConnectionAccess lobjConnection = new
DataConnectionAccess("Database.mdb", "", "", DatabaseManager.Access);
    DataTable ldatData = lobjConnection.SQLSelectExecute("SELECT * FROM
Staff");
    lobjConnection.CloseConnection();

     ListView lobjListView = CreateWindowDataGrid("datListView", ldatData,
600, 300, 100, 150, 0, 0, true);

     MainGrid.Children.Add(lobjListView);

}
```

Según el código fuente anterior, al ejecutar la aplicación, se mostraría en pantalla el siguiente formulario “SwanCSharp”:

11.1.6.4 TabControl

En la clase “WindowBase” disponemos de un control “TabControl” personalizado con el estilo acorde a la ventana “SwanCSharp”. Existe una función que nos devuelve un objeto “TabControl” personalizado: “CreateWindowTabControl”. La función dispone de nueve parámetros: El nombre que va a tener el control, un array de “strings” con los elementos de las pestañas, el ancho del control, el alto del control, los siguientes cuatro parámetros se corresponden al “Margin”, y el último parámetro se indica si se desea un efecto de sombra al control. Un ejemplo de llamada a un control “TabControl” puede ser:

```
string[] lstrItems = new string[3];
lstrItems[0] = "Item 1";
lstrItems[1] = "Item 2";
lstrItems[2] = "Item 3";

TabControl lobjTab = CreateWindowTabControl("tabMain", lstrItems, 350, 300,
40, 80, 40, 40, true);

MainGrid.Children.Add(lobjTab);
```

A continuación se muestra el código fuente completo del ejemplo anterior que muestra un “TabControl” con los datos existentes en una base de datos “Microsoft Access”:

```
public Window1() : base(false)
{
    InitializeComponent();
```

```
    SetWindowTitle("Test of SwanCSharp", Brushes.White);

    SetIconWindowTitle(Properties.Resources.SwanCSharp_Solo_Logo_Grande1);

    ShowCloseButton();
    ShowMaximizeButton();
    ShowMinimizeButton();

    ShowWaterMark(Properties.Resources.SwanCSharp_Nuevo_Grande, 512, 190,
WaterMarkStartPosition.Center, 0, 0, 0, 0);

    IsMoveable(true);

    string[] lstrItems = new string[3];
    lstrItems[0] = "Item 1";
    lstrItems[1] = "Item 2";
    lstrItems[2] = "Item 3";

    TabControl lobjTab = CreateWindowTabControl("tabMain", lstrItems, 350,
300, 40, 80, 40, 40, true);

    MainGrid.Children.Add(lobjTab);

}
```

Según el código fuente anterior, al ejecutar la aplicación, se mostraría en pantalla el siguiente formulario “SwanCSharp”:

11.2 SwanCSharp_Controls.WindowError

La clase "WindowError" nos permite mostrar una ventana de error con el formato y estilo de las ventanas “WindowBase” de este espacio de nombres “SwanCSharp”. Un ejemplo de código fuente que muestra una ventana “WindowError” es el siguiente:

```
WindowError lobjError = new WindowError("Error", "Test of error message.");
lobjError.ShowDialog();
lobjError.Close();
```

Al ejecutar el código fuente del ejemplo anterior se muestra en pantalla la siguiente ventana de información:

Opcionalmente existe un tercer parámetro que nos permite elegir el color principal de la ventana, elegido entre los seis temas existentes.

11.3 SwanCSharp_Controls.WindowInformation

La clase "WindowInformation" nos permite mostrar una ventana de información con el formato y estilo de las ventanas “WindowBase” de este espacio de nombres “SwanCSharp”. Un ejemplo de código fuente que muestra una ventana “WindowInformation” es el siguiente:

```
WindowInformation lobjInformation = new WindowInformation("Information",
"Test of information message.");
lobjInformation.ShowDialog();
lobjInformation.Close();
```

Al ejecutar el código fuente del ejemplo anterior se muestra en pantalla la siguiente ventana de información:

Opcionalmente existe un tercer parámetro que nos permite elegir el color principal de la ventana, elegido entre los seis temas existentes.

11.4 SwanCSharp_Controls.WindowOKCancelQuestion

La clase "WindowOKCancelQuestion" nos permite mostrar una ventana de pregunta con el formato y estilo de las ventanas “WindowBase” de este espacio de nombres “SwanCSharp”. Un ejemplo de código fuente que muestra una ventana “WindowOKCancelQuestion” es el siguiente:

```
WindowOKCancelQuestion lobjQuestion = new
WindowOKCancelQuestion("Question", "We will proceed with copying files.",
SwanCSharp_Controls.InterfaceLanguage.English);
            lobjQuestion.ShowDialog();
if (lobjQuestion.WindowResponse == WindowResult.OK)
{
   /*
   **** The user has selected 'OK'
   */
}
else
{
   /*
   **** The user has selected 'Cancel'
   */
}
lobjQuestion.Close();
```

Al ejecutar el código fuente del ejemplo anterior se muestra en pantalla la siguiente ventana de información:

Opcionalmente existe un cuarto parámetro que nos permite elegir el color principal de la ventana, elegido entre los seis temas existentes.

11.5 SwanCSharp_Controls.WindowWarning

La clase "WindowWarning" nos permite mostrar una ventana de aviso con el formato y estilo de las ventanas “WindowBase” de este espacio de nombres “SwanCSharp”. Un ejemplo de código fuente que muestra una ventana “WindowWarning” es el siguiente:

```
WindowWarning lobjWarning = new WindowWarning("Warning", "Test of warning
message.");
lobjWarning.ShowDialog();
lobjWarning.Close();
```

Al ejecutar el código fuente del ejemplo anterior se muestra en pantalla la siguiente ventana de información:

Opcionalmente existe un tercer parámetro que nos permite elegir el color principal de la ventana, elegido entre los seis temas existentes.

11.6 SwanCSharp_Controls.WindowYesNoQuestion

La clase "WindowYesNoQuestion" nos permite mostrar una ventana de pregunta con el formato y estilo de las ventanas “WindowBase” de este espacio de nombres “SwanCSharp”. Un ejemplo de código fuente que muestra una ventana “WindowYesNoQuestion” es el siguiente:

```
WindowYesNoQuestion lobjQuestion = new WindowYesNoQuestion("Question", "Do
you like SwanCSharp?", SwanCSharp_Controls.InterfaceLanguage.English);
lobjQuestion.ShowDialog();
if (lobjQuestion.WindowResponse == WindowResult.Yes)
{
    /*
    **** The user has selected 'Yes'
    */
}
else
{
    /*
    **** The user has selected 'No'
    */
}
lobjQuestion.Close();
```

Al ejecutar el código fuente del ejemplo anterior se muestra en pantalla la siguiente ventana de información:

Opcionalmente existe un cuarto parámetro que nos permite elegir el color principal de la ventana, elegido entre los seis temas existentes.

11.7 WindowFile

Esta ventana nos mostrará en estructura de árbol todas las unidades lógicas del sistema y sus carpetas asociadas, y sus archivos, permitiendo seleccionar un archivo concreto (cuadro de diálogo para abrir archivos), y devolviendo la ruta completa seleccionada. Para llamar a la ventana de selección de archivos se escribe el siguiente código:

```
WindowFile lobjFile = new WindowFile("", InterfaceLanguage.English);
lobjFile.Owner = this;
lobjFile.ShowDialog();
string lstrFullFilename = lobjFile.FullFilename;
string lstrFile = lobjFile.Filename;
string lstrPath = lobjFile.Path;
lobjFile.Close();
```

La ventana se puede mostrar en español e inglés. En el primer parámetro del constructor se pasa la máscara deseada (si se deja en blanco, se asume *.*) y obtenemos tres propiedades que nos devuelven todos los valores del archivo seleccionado (nombre de fichero completo, nombre de fichero sin ruta, nombre de ruta sin fichero). Un ejemplo de pantalla es:

Opcionalmente existe un tercer parámetro que nos permite elegir el color principal de la ventana, elegido entre los seis temas existentes.

11.8 WindowFolder

Esta ventana nos mostrará en estructura de árbol todas las unidades lógicas del sistema y sus carpetas asociadas, permitiendo seleccionar una carpeta o unidad concreta, y devolviendo la ruta completa seleccionada. Para llamar a la ventana de selección de carpetas se escribe el siguiente código:

```
WindowFolder lobjFolder = new WindowFolder(InterfaceLanguage.English);
lobjFolder.Owner = this;
lobjFolder.ShowDialog();
string lstrPath = lobjFolder.SelectedFolder;
lobjFolder.Close();
```

La ventana se puede mostrar en español e inglés, y según el código de ejemplo en la variable “lstrPath” recibiremos la ruta seleccionada en la ventana. Un ejemplo de pantalla es:

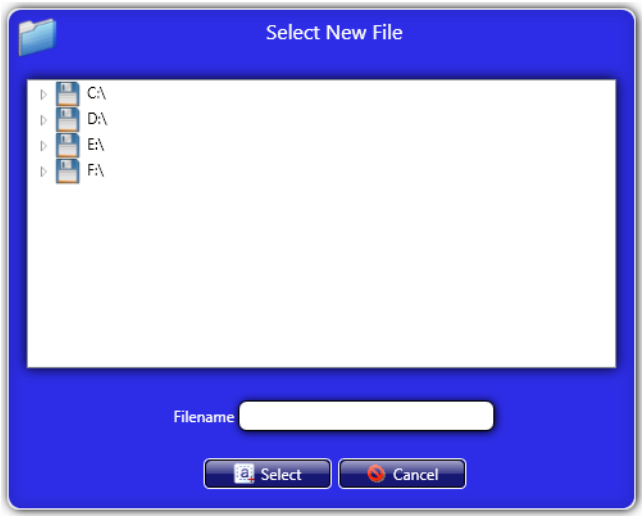
Opcionalmente existe un segundo parámetro que nos permite elegir el color principal de la ventana, elegido entre los seis temas existentes.

11.9 WindowNewFile

Esta ventana nos mostrará en estructura de árbol todas las unidades lógicas del sistema y sus carpetas asociadas, permitiendo construir el “path” de un archivo nuevo concreto (cuadro de diálogo para crear archivos), y devolviendo la ruta completa seleccionada. Para llamar a la ventana de selección de archivos se escribe el siguiente código:

```
WindowNewFile lobjFile = new WindowNewFile("swc",
InterfaceLanguage.English);
lobjFile.Owner = this;
lobjFile.ShowDialog();
string lstrNewFile = lobjFile.File;
lobjFile.Close();
```

La ventana se puede mostrar en español e inglés. En el primer parámetro del constructor se pasa la extensión deseada para el nombre de archivo que el usuario escriba en pantalla (si no se desea extensión alguna, se informa con “String.Empty”); obtenemos una propiedad (File) que nos devuelve la ruta completa construida mediante la ruta seleccionada, el nombre de archivo tecleado en pantalla, y la extensión seleccionada al abrir la ventana. Un ejemplo de pantalla es:



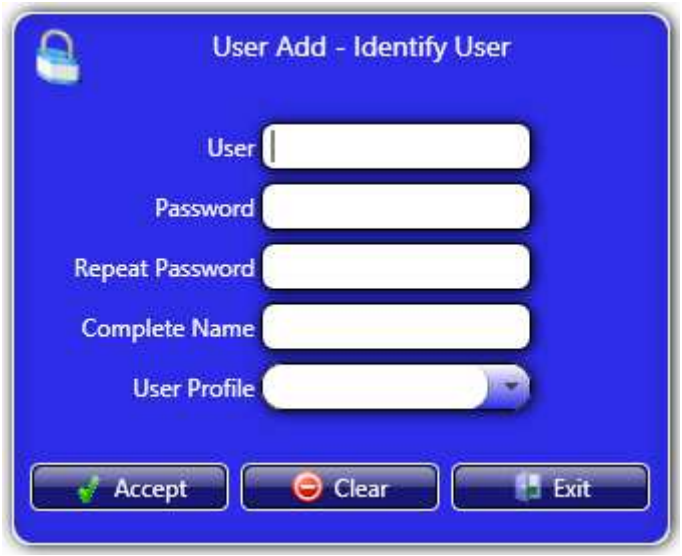
Opcionalmente existe un tercer parámetro que nos permite elegir el color principal de la ventana, elegido entre los seis temas existentes.

11.10 WindowAddUser

Esta ventana “Add User” nos permitirá incorporar una opción para dar de alta nuevos usuarios en un objeto “UserManagement”. Un ejemplo de llamada a la ventana “Add User” es:

```
private void mnuAddUser_Click(object sender, EventArgs e)
{
      WindowAddUser lobjAddUser = new WindowAddUser(ref gobjUserManage,
SwanCSharp_Controls.InterfaceLanguage.English);
      lobjAddUser.ShowDialog();
      lobjAddUser.Close();
}
```

Como primer parámetro se pasa el objeto “UserManagementSQLServer” (pudiendo ser también de Oracle, Firebird, MySQL, o Access), y como segundo parámetro se pasa el idioma de la interfaz gráfica. Opcionalmente existe un tercer parámetro que nos permite elegir el color principal de la ventana, elegido entre los seis temas existentes. Un ejemplo de pantalla es:



Esta pantalla nunca va a permitir a un usuario con perfil “StandarUser” crear otro usuario. A un usuario con perfil “Administrator” solo le permite crear otro usuario “administrador” u otro usuario “StandarUser”. Para el usuario “SuperAdmin” no existen restricciones de ningún tipo.

11.11 WindowLoginUser

Lo lógico es que al ejecutar la aplicación nos muestre una ventana de “login” para verificar la entrada con un usuario correcto. Por lo tanto antes de abrir nuestra ventana principal “Window1” deberemos llamar a la ventana de “Login”. Entonces el código fuente quedaría así para los desarrollos con la ventana “WindowBase” (es necesario referenciar los nombres de espacio SwanCSharp.Users y SwanCSharp_Controls):

```
public UserManagementSQLServer gobjUserManage = null;

public Window1() : base(false)
{
     InitializeComponent();

     SetWindowTitle("Test of SwanCSharp", Brushes.White);

     SetIconWindowTitle(Properties.Resources.SwanCSharp_Solo_Logo_Grande1);

     ShowCloseButton();
     ShowMaximizeButton();
     ShowMinimizeButton();

     IsMoveable(true);

     gobjUserManage = new UserManagementSQLServer("COMPUTER-
NAME\\SQLEXPRESS", "Database");
     WindowLoginUser lobjLogin = new WindowLoginUser(ref gobjUserManage,
SwanCSharp_Controls.InterfaceLanguage.English);
     lobjLogin.ShowDialog();
     lobjLogin.Close();
```

```
     if (!gobjUserManage.LoggedIn)
     {
          WindowError lobjError = new WindowError("Test", "Login
incorrect.");
          lobjError.ShowDialog();
          lobjError.Close();
          Application.Current.Shutdown();
     }
}
```

El constructor de esta ventana tiene una sobrecarga para cada uno de los gestores de bases de datos que soporta la clase "UserManagement" (SQL Server, Oracle, Firebird, MySQL, y Access). Los formularios de Windows de esta clase se pueden mostrar en dos idiomas mediante el enumerado InterfaceLanguage, español e inglés. Opcionalmente existe un tercer parámetro que nos permite elegir el color principal de la ventana, elegido entre los seis temas existentes.

El código anterior mostrará una ventana de login, si el usuario introducido no es correcto, la aplicación se cerrará. A partir de la creación del objeto "UserManagementSQLServer" disponemos de una variable global "gobjUserManage" disponible en toda la aplicación que tiene propiedades con los valores del usuario que ha accedido vía login (UserName, UserPassword, UserCompleteName, Profile).

11.12 WindowPasswordUser

Esta ventana nos permitirá modificar la contraseña del usuario activo que está logueado en ese momento. Este método abrirá una ventana donde se introducirá la contraseña vigente, y se introducirá la nueva contraseña deseada. Al pulsar en "Aceptar" la contraseña será modificada siempre y cuando la contraseña actual coincida con la introducida en pantalla.

Para llamar a la ventana de cambio de contraseña se escribe el siguiente código:

```
WindowPasswordUser lobjPassword = new WindowPasswordUser(ref
gobjUserManage, SwanCSharp_Controls.InterfaceLanguage.English);
lobjPassword.ShowDialog();
lobjPassword.Close();
```

Como primer parámetro se pasa el objeto “UserManagementSQLServer” (pudiendo ser también Oracle, Firebird, MySQL, o Access), y como segundo parámetro se pasa el idioma de la interfaz gráfica. Opcionalmente existe un tercer parámetro que nos permite elegir el color principal de la ventana, elegido entre los seis temas existentes. Un ejemplo de pantalla es:

11.13 WindowRemoveUser

Esta ventana nos permitirá eliminar usuarios de la base de datos. Para eliminar a un usuario únicamente es necesario introducir su nombre. Es importante saber que existen unas reglas para poder borrar usuarios: Si el usuario activo es “SuperUser” puede eliminar a cualquier otro usuario sea cual sea su perfil; si el perfil del usuario activo es “Administrador”, puede eliminar únicamente a usuarios “Standard”. El usuario “Standard” no tiene permisos para borrar a ningún otro usuario sea cual sea su perfil.

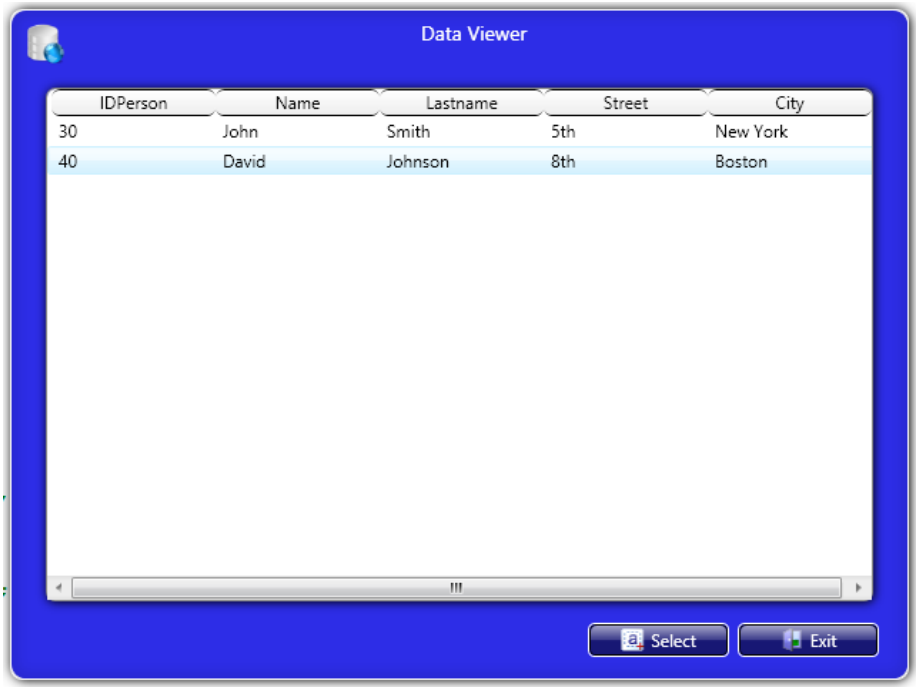
Para llamar a la ventana de borrado de usuarios se escribe el siguiente código:

```
WindowRemoveUser lobjRemove = new WindowRemoveUser(ref gobjUserManage,
SwanCSharp_Controls.InterfaceLanguage.English);
lobjRemove.ShowDialog();
lobjRemove.Close();
```

Como primer parámetro se pasa el objeto “UserManagementSQLServer” (pudiendo ser también Oracle, Firebird, MySQL, o Access) creado para la aplicación, y como segundo parámetro se pasa el idioma de la interfaz gráfica. Opcionalmente existe un tercer parámetro que nos permite elegir el color principal de la ventana, elegido entre los seis temas existentes. Un ejemplo de pantalla es:

11.14 WindowDataQuery

La clase "WindowDataQuery" nos permite mostrar una ventana con un "Grid" de datos y nos devuelve un "DataRow" con la fila que se ha seleccionado en la ventana. Una captura de pantalla es la siguiente:



La clase "WindowDataQuery" espera recibir cuatro parámetros que son los siguientes: el DataTable con los datos; el titulo deseado para el formulario que se va a mostrar; el tamaño de la ventana (mediante el enumerado QueryWindowSize, eligiendo entre Small, Medium, High); y el idioma del formulario (mediante el enumerado InterfaceLanguage, eligiendo entre español e inglés). Opcionalmente existe un quinto parámetro que nos permite elegir el color principal de la ventana, elegido entre los seis temas existentes.

Después de ejecutar la función aparecerá un formulario Windows mostrando los datos en pantalla. El usuario puede seleccionar una línea mediante doble-clic sobre la fila, o un clic

sobre la fila y pulsando en el botón "Seleccionar". La función devolverá un "DataRow" con los datos de la fila elegida.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp_Controls;
```

Un código fuente de ejemplo puede ser:

```
DataConnectionAccess lobjConnection = new
DataConnectionAccess("Database.mdb", "", "", DatabaseManager.Access);
DataTable ldatData = lobjConnection.SQLSelectExecute("SELECT * FROM
Staff");
lobjConnection.CloseConnection();

WindowDataQuery lobjData = new WindowDataQuery(ldatData, "Data Viewer",
QueryWindowSize.Small, SwanCSharp_Controls.InterfaceLanguage.English);
lobjData.ShowDialog();
DataRow ldarRow = lobjData.SelectedRow;
lobjData.Close();
```

11.15 WindowParameters

La ventana de “WindowParameters” nos permitirá visualizar y modificar en pantalla todos los parámetros de configuración asociados a nuestros desarrollos (existiendo cinco constructores, uno por cada gestor de los datos, SQL Server, Oracle, Firebird, MySQL, Access, y fichero externo). La ventana permite mostrarse en dos modos diferentes; un modo para el administrador donde podrá añadir, modificar, y eliminar todos los parámetros (visibles y no visibles); otro modo donde se podrán consultar y modificar únicamente los parámetros visibles, no pudiendo en ningún caso crear uno nuevo o eliminar un parámetro de configuración.

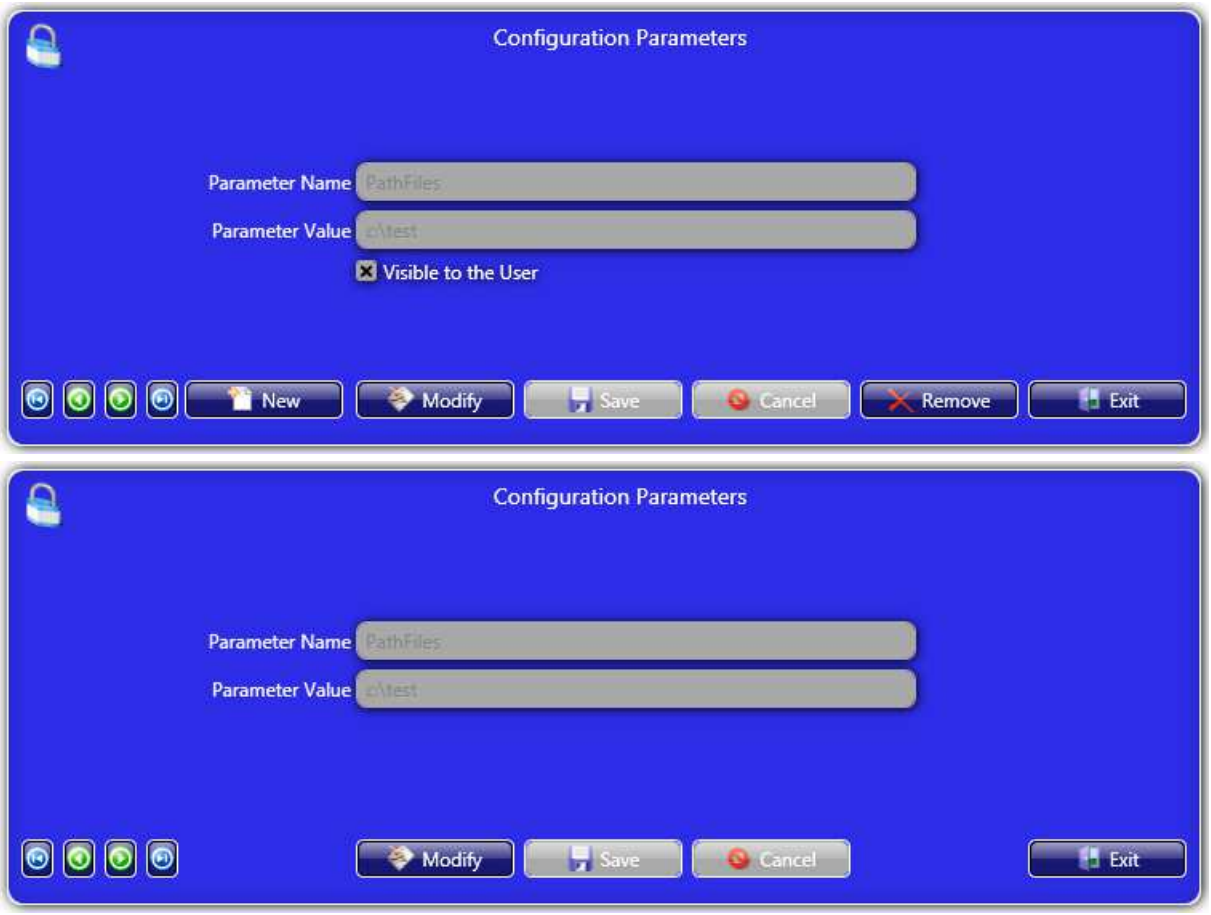
Para llamar a la ventana de modificación de parámetros con todos los permisos podemos escribir el siguiente código:

```
gobjConfig = new ConfiguratorFile("config.cfg");
WindowParameters lobjParameters = new WindowParameters(gobjConfig, false,
true, InterfaceLanguage.English);
lobjParameters.ShowDialog();
lobjParameters.Close();
```

Para llamar a la ventana de modificación de parámetros solo para visualizar y modificar los parámetros “visibles” podemos escribir el siguiente código:

```
gobjConfig = new ConfiguratorFile("config.cfg");
WindowParameters lobjParameters = new WindowParameters(gobjConfig, true,
false, InterfaceLanguage.English);
lobjParameters.ShowDialog();
```

```
lobjParameters.Close();
```



Opcionalmente existe un quinto parámetro que nos permite elegir el color principal de la ventana, elegido entre los seis temas existentes.

11.16 WindowReportView

La clase "WindowReportView" nos permite mostrar en una ventana cualquier informe HTML ya existente, sin necesidad de volver a generarlo. Es necesario informar los siguientes parámetros:

* Nombre del fichero.- El nombre del fichero que tendrá cuando se almacene en disco.
* Ruta del fichero.- La ruta del disco donde se encuentra el fichero html.
* Título ventana.- El título que se va a mostrar en la ventana.
* Tamaño de la ventana de visualización.- Se puede elegir entre Small, Medium, High.
* Idioma del interfaz de pantalla.- Se elige el idioma en el que se va a mostrar la pantalla de visualización del informe.

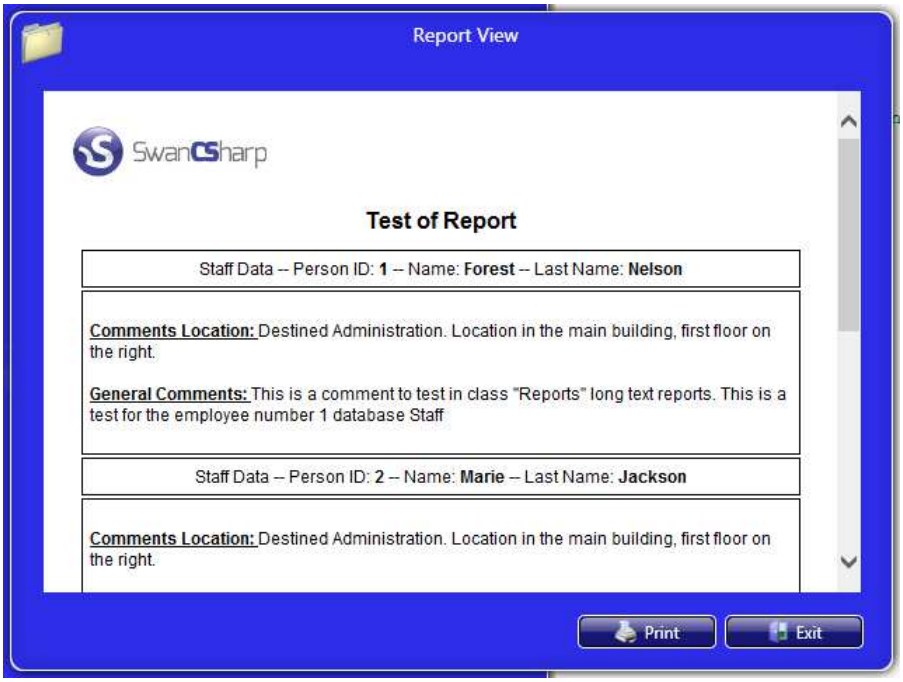
En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp_Controls;
```

Para crear el objeto cliente se utilizan los comandos:

```
WindowReportView lobjReport = new WindowReportView("report.html", "",
"Report View", SwanCSharp.Reporting.ReportViewWindowSize.Small,
InterfaceLanguage.English);
lobjReport.ShowDialog();
lobjReport.Close();
```

El código fuente anterior mostraría la siguiente ventana:



Opcionalmente existe un sexto parámetro que nos permite elegir el color principal de la ventana, elegido entre los seis temas existentes.

11.17 WindowSplash

La clase “WindowSplash” nos permite mostrar en una ventana de bienvenida (splash) al ejecutar nuestra aplicación, o también cualquier ventana que se desee mostrar durante unos segundos. La ventana “WindowSplash” permite mostrar un logotipo principal en el centro de la ventana, un logotipo secundario en cualquiera de las cuatro esquinas, y un texto centrado horizontalmente que se puede ajustar verticalmente. La ventana se puede mostrar en cualquiera de los seis temas de colores existentes. Los parámetros de la ventana son:

* Texto.- Texto a mostrar. Saldrá centrado en la horizontal. Si no se desea texto se informa “vacío”.
* Margen Superior.- La distancia vertical a la que se desea situar el texto.

* Ancho.- Ancho de la ventana “Splash”.
* Alto.- Alto de la ventana “Splash”.
* Imagen principal.- Imagen que se mostrará en el centro de la ventana “Splash”.
* Imagen secundaria.- Imagen secundaria que se mostrará en el centro de la ventana “Splash”. Si no se desea imagen secundaria, se informa con “null”.
* Posición imagen secundaria.- Esquina de la ventana “Splash” donde se mostrará la imagen secundaria.
* Tiempo hasta cierre.- Tiempo (en segundos) que se mantendrá abierta la ventana “Splash”, hasta su cierre automático.
* Transparencia.- Si se desea o no la transparencia en la ventana “Splash”.
* Color (Tema).- Color principal de la ventana “Splash”.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp_Controls;
```

Para crear la ventana “Splash” se utilizan los comandos:

```
WindowSplash lobjSplash = new WindowSplash("Loading...", 250, 600, 400,
Properties.Resources.Solariem, Properties.Resources.Powered_SwanCSharp,
PositionSecondaryImageSplash.BottomRightCorner, 6, true, WindowTheme.Blue);
lobjSplash.ShowDialog();
lobjSplash.Close();
```

Para ejecutar la ventana de bienvenida justo antes del arranque de la aplicación, se recomienda situar el código fuente anterior dentro del constructor de la ventana principal de nuestra aplicación. El código fuente anterior mostraría la siguiente ventana:

11.18 SwanCSharp_Controls.ClipboardAgent

La clase ClipboardAgent nos permite capturar todo aquello que sea copiado o cortado en el portapapeles de Windows.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp_Controls;
```

Para crear el objeto “ClipboardAgent” se realiza lo siguiente:

```
ClipboardAgent lobjClipAgent = new ClipboardAgent(this);

lobjClipAgent.ClipboardReceiveText += new
ClipboardAgent.ClipboardReceiveTextEventHandler(lobjClipAgent_ClipboardRece
iveText);

lobjClipAgent.ClipboardReceiveAudio += new
ClipboardAgent.ClipboardReceiveAudioEventHandler(lobjClipAgent_ClipboardRec
eiveAudio);

lobjClipAgent.ClipboardReceiveFiles += new
ClipboardAgent.ClipboardReceiveFilesEventHandler(lobjClipAgent_ClipboardRec
eiveFiles);

lobjClipAgent.ClipboardReceiveImage                    +=                    new
ClipboardAgent.ClipboardReceiveImageEventHandler(lobjClipAgent_ClipboardRec
eiveImage);
```

No solo estamos creando el objeto, sino también estamos creando un evento por cada tipo de datos que el portapapeles es capaz de capturar, en total cuatro eventos, por lo tanto hay que crear las subrutinas para cada evento:

```
private void lobjClipAgent_ClipboardReceiveImage(ClipboardAgent.ContentType
penuContentType, BitmapSource pobjImage)
{

}

private void lobjClipAgent_ClipboardReceiveFiles(ClipboardAgent.ContentType
penuContentType, System.Collections.Specialized.StringCollection pstrFiles)
{

}

private void lobjClipAgent_ClipboardReceiveAudio(ClipboardAgent.ContentType
penuContentType, Stream pobjAudio)
{

}
```

```
private void lobjClipAgent_ClipboardReceiveText(ClipboardAgent.ContentType
penuContentType, string pstrText)
{

}
```

En la parte deseada de nuestro código fuente iniciamos el agente:

```
lobjClipAgent.StartAgent();
```

Al cerrar nuestra aplicación, o en la parte de nuestro software que se estime oportuno, se cierra el agente:

```
lobjClipAgent.CloseAgent();
```

11.19 SwanCSharp_Controls.GlobalHotKeys

La clase GlobalHotKeys nos permite gestionar teclas de acceso rápido a nivel global (sistema operativo). Podemos asignar en el sistema operativo una combinación de teclas y después ejecutar un proceso cada vez que se pulsen.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp_Controls;
```

Para crear el objeto “GlobalHotKeys” se realiza lo siguiente:

```
GlobalHotKeys mobjHotKey = new GlobalHotKeys(this);

mobjHotKey.HotKeyIsPressed += new
GlobalHotKeys.HotKeyPressedEventHandler(mobjHotKey_HotKeyIsPressed);

mobjHotKey.StartHotKey(Key.LeftAlt, 0x42);
```

No solo estamos creando el objeto, sino también estamos creando un evento que saltará cuando sean pulsadas las teclas. Mediante el método “StartHotKey” iniciamos la escucha según los parámetros pasados. En el primer parámetro determinamos si deseamos pulsación de tecla auxiliar (pueden ser `Key.LeftAlt`, `Key.LeftCtrl`, `Key.RightCtrl`, `Key.LeftShift`, `Key.RightShift`), y en el segundo parámetro el valor hexadecimal ASCII de la tecla deseada. En el ejemplo pasamos “0x42” ya que el valor ASCII en hexadecimal 42, es el valor decimal 66, que se corresponde a la letra “B”. Por lo tanto en el ejemplo anterior estamos controlando las pulsaciones de la combinación “Alt+B”. Si no deseamos incluir una tecla auxiliar, debemos informar el primer parámetro con valor “0”.

El último paso es crear el método que será llamado por el evento de la clase:

```
private void mobjHotKey_HotKeyIsPressed(Key pobjAuxKey, int pintKey)
```

```
{
}
```

Al cerrar nuestra aplicación, o en la parte de nuestro software que se estime oportuno, se paraliza el objeto HotKey:

```
mobjHotKey.StopHotKey();
```

11.20 SwanCSharp_Controls.SW_Miscellaneous

La clase SW_Miscellaneous tiene como finalidad englobar aquellas funciones genéricas existentes en SwanCSharp.Miscellaneous que necesitan una personalización para poder ser ejecutadas en una ventana SwanCSharp.WindowBase.

11.20.1 LoadDataInComboBox

Este procedimiento nos permite cargar en un ComboBox (generado desde la clase WindowBase) todos los elementos deseados, insertando en cada elemento los parámetros Text y Value deseados (algo que no permite directamente el objeto ComboBox). Para ello se va a crear un array de string con el parámetro “Text” de cada elemento, y otro array de “String” con el parámetro “Value” de cada elemento. En el tercer parámetro se pasa por referencia el “ComboBox” que deseamos cargar.

En la cabecera del procedimiento se añade la referencia a la clase:

```
using SwanCSharp_Controls;
```

Un código de ejemplo puede ser:

```
private void Form1_Load(object sender, EventArgs e)
{
  string[] lstrData = new string[3];
  string[] lstrValue = new string[3];

  lstrData[0] = "Option 1";
  lstrValue[0] = "1";

  lstrData[1] = "Option 2";
  lstrValue[1] = "2";

  lstrData[2] = "Option 3";
  lstrValue[2] = "3";

  ComboBox lobjComboBox = CreateWindowComboBox("cmbComboBox", 150, 24, 100,
150, 0, 0, true);

  Miscellaneous.LoadDataInComboBox(lstrData, lstrValue, ref lobjComboBox);
}
```