Functions library for .Net Framework 2.0/3.0 or higher

SwanCSharp v4.5

FreeWare License

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

v4.5 (January 2014)

REVISION CONTROL

VERSION	DATE	COMMENTS
4.5 Rev1	02/01/2014	Initial documentation

DOCUMENT INDEX

1. ACKNOWLEDGEMENTS

Thanks to Pedro Pablo Fernández (http://www.pedropablofernandez.com), for designing the SwanCSharp logo.

2. SwanCSharp in Internet

The "SwanCSharp" project exists in Internet on the official website and on social networks such as Facebook and Twitter. Here are the addresses of Internet access and QR codes for easy connection through mobile.

2.1 Oficial Web

In the official Web SwanCSharp can download any version of the project and you can download the documentation in Spanish or English. There is also a form to communicate via email with project managers.

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

QR Code:

2.2 Facebook

In the social network Facebook there is an official page dedicated to the project SwanCSharp which aims to create a community of users of this project.

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

QR Code:

2.3 Twitter

In the social network Twitter there an official page dedicated to SwanCSharp project which aims to create a community of users of this project.

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

Account: **@swancsharp**

QR Code:

3. END USER FREEWARE LICENSE AGREEMENT

USER WARNING: *Please read carefully. Using all or a portion of the Software is accepting all the terms and conditions of this Agreement. If you do not agree, do not use this Software.*

3.1 DEFINITIONS

Under this Agreement, the following terms shall have the respective meanings indicated; such meanings apply to both singular and plural forms of the terms defined:

"Licensor" means http://www.swancsharp.com and Manuel Llaca as product developer.

"Licensee" means You or Your Company, unless otherwise indicated.

"Software" means (a) all the contents of the files, disk (s), CD-ROM (s) or any other means which this Agreement is provided, including but not limited to ((i) registration information, (ii) related explanatory materials or files ("Documentation"), and (iii) configuration files, execution and code examples of Software (if any)), and (b) upgrades, modified versions, additions, and

copies of the Software, any, licensed to you by http://www.swancsharp.com (collectively, "Updates").

"Using" "Use" or "Using" means to access, install, download, copy or other benefits obtained from using the functionality of the Software in accordance with the Documentation.

"System" means XO, Windows OS, GNU / Linux or Mac OSX, or any other virtual machine.

3.2 GENERAL PURPOSE

It grants you a license to use the Software nonexclusive downloaded for any purpose for unlimited period. The software product under this License is provided free. While monetary transaction is not effected by the license to use this software that does not mean that there are no conditions for using such software:

- The Software may be installed and used by the Licensee for any legal purpose.

- The Software may be installed and used by the Licensee on any number of systems.

- The Software may be copied and distributed under the condition that the copyright and warranty are intact, and the dealer does not charge money or fees for the Software product, except cover distribution costs.

- Licensee may reference the Software to its commercial projects, and can earn money on its commercial software without paying royalty to Licensor for reference the Software.

- Licensee shall have no right or ownership in the Software. Licensee acknowledges and agrees that the Licensor retains all copyrights and all other rights, including the right to property, and to the Software.

- Use in the scope of this License is royalty free and no license or registration shall be paid by the Licensee.

3.3 INTELLECTUAL PROPERTY RIGHTS

- This License does not transmit any intellectual rights on the Software. The Software and any copies that Licensee is authorized by Licensor to make are the intellectual property of Licensor and its suppliers, and belong.

- The Software is protected by copyright, including without limitation copyright laws and international treaty provisions.

- Any copies that the Licensee is permitted to make pursuant to this Agreement must contain the same copyright and other proprietary notices that appear on or in the Software.

- The structure, organization and code of the Software are trade secrets and confidential information of Licensor and its suppliers. Licensee agrees not to decompile, disassemble or otherwise discover any source code of the Software.

- Any attempt to reverse engineer, copy, clone, modify or alter in any way the installer program without the Licensor specific approval are strictly prohibited. The Licensee is not authorized to use any plug-in or attached to allow saving changes to a file with software licensed and distributed by the Licensor.

- Any information provided by Licensor or obtained by Licensee, as then permits may only be used by the Licensee for the purpose described herein and may not be disclosed to any third party or used to create any software which is substantially similar to that expressed by Software.

- The marking must be used in accordance with accepted trademark practice, including identification of the trademark owners. Trademarks may only be used to identify any printed output of the Software and such use of trademarks does not grant Licensee any ownership of and for them.

3.4 WARRANTY

Licensor warrants:

- Http://www.swancsharp.com has ownership in the Software and its documentation and / or is in possession of valid rights and licenses that support the terms of this Agreement.

- To the best knowledge and belief of the Licensee, the Software will not infringe or violate any intellectual property right of any third party.

- The Software does not contain any back door, time bomb, make suicide device or other routine intentionally designed by the Licensor to disable a computer program, or instructions that alter, destroy or inhibit the processing environment.

- Except those warranties specified in section 1.4 above, the Software is being delivered "HOW IS" and the Licensor makes no warranty as to its use or performance.

- The Licensor and its suppliers do not and can not warrant the performance or results the Licensee may obtain by using the Software. The risk arising out of use or performance of the Software is assumed by the Licensee.

- Licensor makes no warranties, express or implied, that (i) the Software will be of satisfactory quality, fit for any particular purpose or for any particular use within specific conditions, notwithstanding that such purpose, use or condition can be known by Licensor, or (ii) that the Software will operate error free or without interruption or that any errors will be corrected.

3.5 LIMITATION OF RESPONSIBILITIES

In no event shall Licensor or its suppliers be liable for any damages, claims or whatever cost or any consequential, indirect or incidental damages, or any loss, even if the Licensor has been advised of the possibility of such loss, damage, claim or cost made by any third party.

In no event shall Licensee be liable to the Licensor on condition that the Licensee complies with all terms and conditions of this License.

3.6 NO-RESCISSION CLAUSES

If any portion of this agreement is deemed unenforceable, the remainder will remain valid. This means that if a section of the Agreement is not permitted by law, the rest of the Agreement remains in effect. The non-exercise of any right under this Agreement by either party shall not constitute a waiver of (a) any other term or condition of this Agreement, or (b) a right at any time thereafter to enforce accurate and strict terms of this Agreement.

4. INTRODUCTION

SwanCSharp Function Library aims to collect lots of functions to implement development environments based on technology. NET Framework, and these functions have the following objectives: to develop applications faster; put the hand of the beginning programmers develop highly complex processes.

The function library is SwanCSharp fast implementation and ease of use, the entire system only consists of a file called "SwanCSharp.dll".

5. REQUIREMENTS

To develop applications using the functions of "SwanCSharp.dll" requires a programming environment to compile under the Framework. NET 2.0 or higher. Applications can be developed in C # and Visual Basic. Net development environments using Microsoft's IDE (Visual Studio or Free Express versions) or use other IDEs like Netbeans, SharpDevelop or MonoDevelop for Windows, or MonoDevelop for Linux.

All features included in "SwanCSharp.dll" are usable in any application to develop according to the requirements defined in the previous paragraph (Windows Forms application, Windows Service, Web Service, Console Application, Class Library application, etc).

6. Solariem (Graphics wizard for SwanCSharp)

The library "SwanCSharp" has a wizard application called "Solariem" that allows us to generate the source code and entire base of the solution (.SLN) with data access, user

management, parameter management, main screen, icons, etc, ready to run in Visual Studio 2005 Extensions Wpf + or higher.

For more information, visit the official website (http://www.solariem.com). With Solariem can create the basis for an application in C# + SwanCSharp in less than 10 minutes. On the following link you can see a demo video:

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

7. DOWNLOAD AND INSTALLATION

The library "SwanCSharp" can be downloaded from the official website of the project (http://www.swancsharp.com), obtaining a ZIP file called "SwanCSharp.zip". Unpack the ZIP file to the desired location on your hard disk.

Once unzipped the file in the folder "**Binaries**" finds "**SwanCSharp.dll**" file to be copied to the build directory of the Visual Basic or C # in which we want to use the library functions. In the development environment (IDE) usual work creates a new project, or opens an existing project, and included "SwanCSharp.dll" in the reference list. From that moment can be used in this project all library functions.

In the folder "Documentation" you can find the instruction manual in Spanish and English.

In the folder "Samples" you can find many examples of using the features built into the library. All examples have been developed in the IDE Microsoft Visual C #.

8. NEW FEATURES IN 4.5 VERSION

In this section are commented the "SwanCSharp" new features in this new version 4.5 with regard to the previous 4.0 version.

8.1 CheckInternetConnection

In the "Internet" class we added the "CheckInternetConnection" function that returns whether or not Internet connection.

8.2 CheckTCPPortIsOpen

In the “Network” we added the “CheckTCPPortIsOpen” function that returns if a concrete port of a domain name or IP is open.

8.3 ClipboardAgent

In the "SwanCSharp_Controls" namespace we have created a class with a clipboard agent for use in "WindowBase" windows.

8.4 GetDomainNameFromIP

In the "Internet" class we added the "GetDomainNameFromIP" function that returns the domain name of given valid IP.

8.5 GetFileFromHttp

In the "Internet" class we added the "GetFileFromHttp" function that allows us to download a file uploaded to the Internet.

8.6 GetIPFromDomainName

In the "Internet" class we added the "GetIPFromDomainName" function that returns the IP of a given Internet domain name.

8.7 GetIPNetworkData

In the "Network" class we added the "GetIPNetworkData" function that returns all the information associated with the system's Ethernet cards.

8.8 GetListOfCountries

In the "Miscellaneous" class we added the "GetListOfCountries" function that returns a list of countries (in English or Spanish).

8.9 GetWebProxyActive

In the "Internet" class we added the "GetWebProxyActive" function that allows us to obtain all the information of the active web proxy in the execution computer.

8.10 GlobalHotKeys

In the "SwanCSharp_Controls" namespace we have created a class, you can set global hotkeys for use in "WindowBase" windows.

8.11 IsLocalIP

In the "Network" class we added "IsLocalIP" function that checks if a given IP is local or not.

8.12 IsValidIP

In the "Network" class we added the "IsValidIP" function that checks if a given IP address has the correct structure.

8.13 MySQL

To SwanCSharp we have added support for MySQL manager in managing databases, now Access, SQL Server, Oracle, Firebird, and MySQL are supported.

8.14 Network

We have created a new class called "Network" that collects functions and methods for managing computer networks.

8.15 ShowNotifyButton

In the GUI "SwanCSharp_Controls" we have created a new notification button which accompanies to minimize, maximize, and close buttons.

8.16 WindowRemoveUser

We have repaired an existing error validating current user permissions to delete an account.

9. LIBRARY STRUCTURE

The library "**SwanCSharp**" consists of a single DLL (**SwanCSharp.dll**) that containing several namespaces.

9.1 SwanCSharp namespace

The namespace "**SwanCSharp**" is divided into groups of classes which are (**you can use this namespace with the. Net Framework 2.0 or higher installed**):

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

SwanCSharp.Arrays.- Class that adds features and methods for managing arrays, thus avoiding the deficiencies of that class in C #.

SwanCSharp.Configurator.- Class that incorporates to our developments complete management of configuration parameters (including forms).

SwanCSharp.CRC32.- Class that incorporates to our developments CRC32 calculation functions on strings and files.

SwanCSharp.DataAccess.- Class that allows us to incorporate into our developments complete management of data access, including management in the creation and maintenance of the databases.

SwanCSharp.Directories.- Class that contains functions for managing directories and folders.

SwanCSharp.Encryption.- Class that contains functions for data or files encryption of any type (AES256 encryption).

SwanCSharp_Controls.WindowFolder.- Class that incorporates the window basic design of folders selection.

SwanCSharp.Files.- Class that contains functions for file management and maintenance.

SwanCSharp.Imaging.- Class that contains functions for managing and processing images.

SwanCSharp.Internet.- Class that contains functions related to the Internet world (FTP Client, etc..).

SwanCSharp.Logger.- Class that allows us to incorporate to our management developments Log Files exception handling.

SwanCSharp.Miscellaneous.- Generic class which includes those functions and broad general use.

SwanCSharp.Network.- Class that contains functions related to manage computer networks.

SwanCSharp.Reporting.- Class that adds features and methods for easily and quickly develop reports in HTML data for later viewing or printing by integrated viewer.

SwanCSharp.SNTP.- Class that adds features and methods to incorporate our developments SNTP time client.

SwanCSharp.Sockets.- Class that adds features and methods for quickly and easily develop a Client Socket or Server Socket, or both, allowing create applications to perform to enable a client / server communication using any connection (LAN, WAN or Internet) and using TCP port desired.

SwanCSharp.UserManagement- Class that incorporates to our developments complete management of users and profiles (including forms). There is a specific class for each database manager (SQLServer, Oracle, Firebird, and Access).

SwanCSharp.Validations.- Class that allows our developments incorporate functions to validate data.

SwanCSharp.Video.- Class for managing generated video streaming from IP video cameras.

9.2 SwanCSharp_Controls namespace

The namespace "**SwanCSharp_Controls**" is divided into groups of classes which are (**you can use this namespace with the. Net Framework 3.0 or higher installed**):

SwanCSharp_Controls.ClipboardAgent.- We have created this class to include an agent that "watches" for the movement of content in the Windows clipboard.

SwanCSharp_Controls.GlobalHotKeys.- We have created this class to include a global management hotkeys.

SwanCSharp_Controls.SW_Miscellaneous.- We created this generic class which includes those existing functions in SwanCSharp.Miscellaneous that require customization for SwanCSharp.WindowBase use.

SwanCSharp_Controls.WindowBase.- Class that incorporates the basic design of all windows that can be created with this program.

SwanCSharp_Controls.WindowAddUser.- Class that incorporates the basic design of the user creation window.

SwanCSharp_Controls.WindowDataQuery.- Class that incorporates the basic design of display data window.

SwanCSharp_Controls.WindowError.- Class that incorporates the basic design of the message window of error.

SwanCSharp_Controls.WindowFile.- Class that incorporates the basic design of the file selection window.

SwanCSharp_Controls.WindowFolder.- Class that incorporates the basic design of folders selection window.

SwanCSharp_Controls.WindowInformation.- Class that incorporates the basic design of the message window of information.

SwanCSharp_Controls.WindowLoginUser.- Class that incorporates the basic design of the user access window.

SwanCSharp_Controls.WindowNewFile.- Class that incorporates the basic design of the new file selection window (dialog box for new files).

SwanCSharp_Controls.WindowOKCancelQuestion.- Class that incorporates the basic design of the message window of question.

SwanCSharp_Controls.WindowParameters.- Class that incorporates the basic design of configuration parameters.

SwanCSharp_Controls.WindowPasswordUser.- Class that incorporates the basic design of the password change of users.

SwanCSharp_Controls.WindowRemoveUser.- Class that incorporates the basic design of the remove user window.

SwanCSharp_Controls.WindowReportView.- Class that incorporates the basic design of the report view window.

SwanCSharp_Controls.WindowSplash.- Class that incorporates the basic design of the welcome window.

SwanCSharp_Controls.WindowWarning.- Class that incorporates the basic design of the message window of warning.

SwanCSharp_Controls.WindowYesNoQuestion.- Class that incorporates the basic design of the message window of question.

10. FUNCTIONS USER REFERENCE (SwanCSharp)

The following will describe the reference of use of each of the functions incorporated into the SwanCSharp namespace of the library. The references are grouped by the classes to which they belong in alphabetical order.

10.1 SwanCSharp.Arrays

The Arrays class incorporates a number of functions to support the management of different types of arrays. This class is useful for resizing that objects.

10.1.1 AddByteToArray

The function allows to add one byte to the end or beginning of a given byte array, resizing the original array. Expects three parameters: the first parameter in the source array in the second parameter byte to add, and the third parameter a Boolean, true inserts it at the beginning of the array, false inserts it at the end of the array.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

```
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

The function allows you to add an array of bytes source the content of another array of bytes secondary. Expects three parameters: the first parameter in the source array in the second parameter the secondary array, and the third parameter a Boolean, true inserts it at the beginning of the array, false inserts it at the end of the array.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

```
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

The function allows you to add an array of integer source the content of another array of integer secondary. Expects three parameters: the first parameter in the source array in the second parameter the secondary array, and the third parameter a Boolean, true inserts it at the beginning of the array, false inserts it at the end of the array.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

```
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

The function allows you to add an array of objects source the content of another array of objects secondary. Expects three parameters: the first parameter in the source array in the second parameter the secondary array, and the third parameter a Boolean, true inserts it at the beginning of the array, false inserts it at the end of the array.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

```
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

The function allows you to add an array of strings source the content of another array of strings secondary. Expects three parameters: the first parameter in the source array in the second parameter the secondary array, and the third parameter a Boolean, true inserts it at the beginning of the array, false inserts it at the end of the array.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

```
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

The function allows to add one integer to the end or beginning of a given integer array, resizing the original array. Expects three parameters: the first parameter in the source array in the second parameter integer to add, and the third parameter a Boolean, true inserts it at the beginning of the array, false inserts it at the end of the array.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

```
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

The function allows to add one object to the end or beginning of a given object array, resizing the original array. Expects three parameters: the first parameter in the source array in the second parameter object to add, and the third parameter a Boolean, true inserts it at the beginning of the array, false inserts it at the end of the array.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

```
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

The function allows to add one string to the end or beginning of a given string array, resizing the original array. Expects three parameters: the first parameter in the source array in the second parameter string to add, and the third parameter a Boolean, true inserts it at the beginning of the array, false inserts it at the end of the array.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

```
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

The purpose of the function "ArrayResize" is array resizing using a single function more generic.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.1.10 FindItemInByteArray

The function allows search a byte within an array of bytes. Returns true if found and returns false if not found.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.1.11 FindItemInIntegerArray

The function allows search an integer within an array of integers. Returns true if found and returns false if not found.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.1.12 FindItemInObjectArray

The function allows search an object within an array of objects. Returns true if found and returns false if not found.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.1.13 FindItemInStringArray

The function allows search a string within an array of strings. Returns true if found and returns false if not found.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.1.14 RemoveItemsFromByteArray

The function allows you to remove a sequential number of elements in an array of bytes. Expects three parameters: the first parameter in the source array in the second parameter the starting index element to be deleted, and the third parameter the final index of deleted item. The function returns an array of bytes with deleted items ranging from beginning to end.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

```
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

The function allows you to remove a sequential number of elements in an array of integer. Expects three parameters: the first parameter in the source array in the second parameter the starting index element to be deleted, and the third parameter the final index of deleted item. The function returns an array of integer with deleted items ranging from beginning to end.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

```
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

The function allows you to remove a sequential number of elements in an array of objects. Expects three parameters: the first parameter in the source array in the second parameter the starting index element to be deleted, and the third parameter the final index of deleted item. The function returns an array of bytes with deleted items ranging from beginning to end.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

```
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

The function allows you to remove a sequential number of elements in an array of strings. Expects three parameters: the first parameter in the source array in the second parameter the starting index element to be deleted, and the third parameter the final index of deleted item. The function returns an array of strings with deleted items ranging from beginning to end.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

```
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

The class "Configurator" allows us to create a complete system management settings in the applications developed.

10.2.1 Configurator

The class constructor "Configurator" requires support for storing configuration data, and can select from SQL Server, Oracle, Firebird, Access, MySQL, or a standard text file. There is a specific class for each database manager (ConfiguratorSQLServer, ConfiguratorOracle, ConfiguratorFirebird, ConfiguratorAccess, ConfiguratorMySQL, and ConfiguratorFile). The purpose of these classes is to incorporate complete management configuration parameters for our application developed (including management windows). So every time you build the object "Configurator", this class will check our database for table exists "AppConfig" and if it has the necessary fields. Otherwise the first thing it will automatically create all the necessary structure in our database chosen (a process that only runs the first time and without developer intervention). In the event that the option chosen to host the configuration file is written to disk, the builder will create a folder "Config" directory hanging execution of the application and there will create the configuration file (parameter passed in the constructor).

In the case that the chosen data support is SQL Server, the data in the table "AppConfig" will be encrypted to anyone externally to our developments can access.

In the case that the chosen database is Access, the data is not encrypted, but users are recommended to protect the database password (from the Microsoft Access Security option).

In the case that the support option is a configuration file, the content is encrypted.

In all other cases (Oracle, Firebird, and MySQL) the data is not encrypted, so it is recommended to use authentication methods that include their own database managers.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

It is also important to inform the classes "Configurator" will be used throughout the development of our application and therefore it is recommended to create the object using a global variable for configuration data, and calls to the management functions, are accessible from anywhere in the application.

To create the configurator on SQL Server:

```
public ConfiguratorSQLServer gobjConfig;

private void Form1_Load(object sender, EventArgs e)
{
  gobjConfig = new ConfiguratorSQLServer("COMPUTER\\INSTANCE", "DataBase");
}
```

To create the configurator on Oracle:

```
public ConfiguratorOracle gobjConfig;

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

To create the configurator on 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 );
}
```

To create the configurator on 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 );
}
```

To create the configurator on Access:

```
public ConfiguratorAccess gobjConfig;

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

To create the configurator on disk file:

```
public ConfiguratorFile gobjConfig;

private void Form1_Load(object sender, EventArgs e)
{
  gobjConfig = new ConfiguratorFile("filename.cfg");
}
```

10.2.1.1 AddNewConfigParameter

Used to create a new configuration parameter. Requires three parameters, one to indicate which name will be the new parameter, the second to indicate the value of this parameter, the third parameter to indicate whether it will be visible to the user or only for the internal use of the application. The first two parameters are of type "String". For example:

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

10.2.1.2 ExistConfigParameter

Used to check if a parameter exists in the current configuration. Just need a parameter that corresponds to the parameter name to look for. For example:

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

10.2.1.3 GetConfigParameterValue

Used to retrieve the value of an existing parameter. Just need a parameter that corresponds to the parameter name to look for. For example:

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

10.2.1.4 GetDataTableFromAppConfig

Used to get a DataTable with all data related to the parameters. You receive a DataTable containing the following Fields: IDParameter, CF_Name_Parameter, CF_Value_Parameter and CF_Visible_Parameter. An example use of the function is:

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

10.2.1.5 GetDataTableFromAppConfigOnlyVisible

Used to get a DataTable with all data related to the parameters, but only obtain marked as "Visible". You receive a DataTable containing the following Fields: IDParameter, CF_Name_Parameter, CF_Value_Parameter and CF_Visible_Parameter. An example use of the function is:

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

10.2.1.6 RemoveConfigParameter

Used to remove an existing parameter. Only requires one parameter corresponding to the parameter name to remove. For example:

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

10.2.1.7 ShowConfigurationWindow

The "Configuration Window" will allow us to view and modify display all configuration parameters associated with our developments. The window can be displayed in two ways, one way for the administrator where you can add, modify, and delete all parameters (visible and invisible) which otherwise may query and modify only visible parameters, can not in any case create a new or delete a configuration parameter.

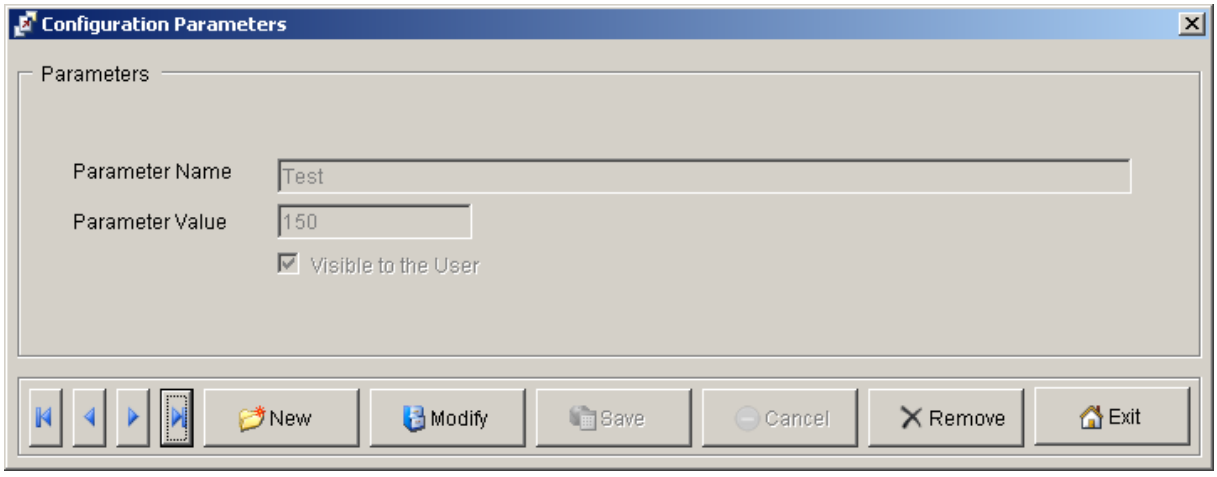
As a suggestion we can say that the type of configuration can be combined with the management class SwanCSharp users, so that we can create a MenuStrip two options, an option that is enabled only for users Superuser profiles (developer application) for all permits access to the configuration parameters. Another option is enabled only for administrators (privileged user configuration); you can modify the values of those parameters "visible". The third single user profile will not have access to the settings window.

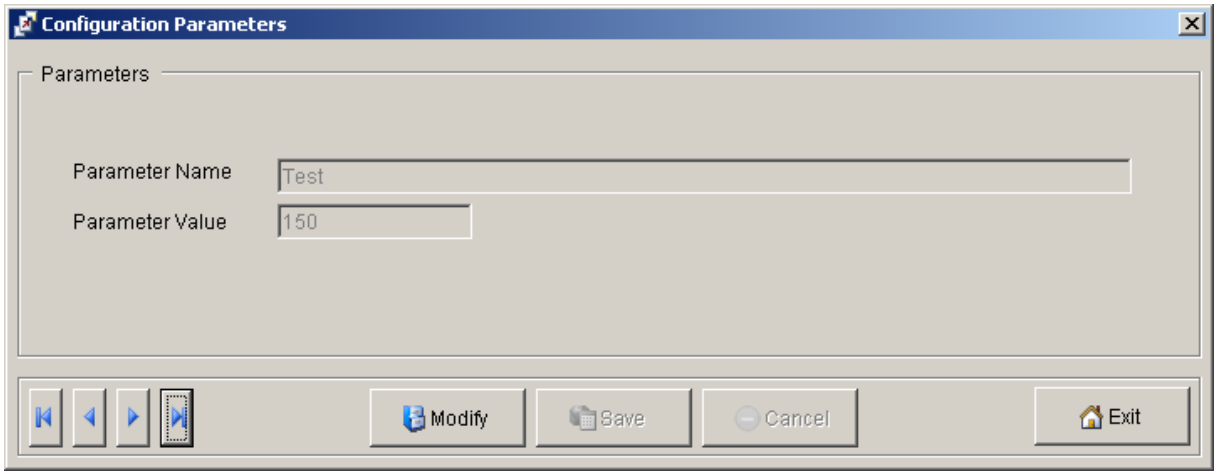
To call the parameter changes window with all the permissions we can write the following code:

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

To call the parameter changes window to view and modify only the parameters "visible" we can write the following code:

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





Important note: In the case of custom forms "SwanCSharp" created from the namespace "SwanCSharp_Controls", you must use the class "WindowParameters" then existing in that namespace instead of this function "ShowConfigurationWindow".

10.2.1.8 UpdateConfigParameterValue

Used to update the value of an existing parameter. Requires three parameters, the first to indicate the name of the parameter to find, the second to specify the new value of the parameter, and the third parameter to specify whether the user will be visible only for internal use or application. For example:

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

10.3 SwanCSharp.CRC32

The class "CRC32" allows us to calculate the value of CRC (32 bits) of any given file, or you can calculate the CRC of a given text string (there are 2 overloads). This function is useful for checking if a file has changed in structure, for example, can be used to verify that an original file executable (EXE) has not been modified at any time, and protect any malicious changes.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

The class constructor "CRC32" expected in the first parameter file path, and the second parameter in the file name.

10.4 SwanCSharp.DataAccess

The DataAccess class allows us to manage databases, supporting SQL Server, Oracle, Firebird, and Access, existing four DataConnection classes (DatabaseConnectionSQLServer, DatabaseConnectionOracle, DatabaseConnectionFirebird, DatabaseConnectionMySQL, DatabaseConnectionAccess). Those classes run through the implementation of standard SQL statements, and has a function to create a connection to the database, another function to close the connection, and two command execution functions, one to run a SELECT, and one for execute INSERT, UPDATE, DELETE, or remaining commands.

Are also available four secondary classes (DatabaseStructureSQLServer, DatabaseStructureOracle, DatabaseStructureFirebird, DatabaseStructureMySQL, and DatabaseStructureAccess) for creating and updating databases SQL Server, Oracle, Firebird, MySQL, or Access dynamically at runtime, allowing you to create applications that automatically create and update databases data.

10.4.1 DataConnectionSQLServer, DataConnectionOracle, DataConnectionFirebird, DataConnectionMySQL, and DataConnectionAccess

There are four types of connection (SQLServer, Oracle, Firebird, MySQL, and Access). Thus DataAccess four classes can be used for making connections simpler and more portable locally or remotely.

To use DataConnectionSQLServer class requires access to the system to a version "Commercial" or "Express" (free) SQL Server, either local (installed on the same machine) or remote access (installed on another computer).

To use DataConnectionOracle class requires access to the system "Commercial" or "Express" version (free) from Oracle, either local (installed on the same machine) or remote access (installed on another computer). On the computer that is to be executed the application using DataConnectionOracle is necessary to install the .Net Oracle software connection called ODP .Net (a valid engine version for Oracle database is installed on the system).

To use DataConnectionFirebird class requires access to the system a version of Firebird, as well as copy along with "SwanCSharp.dll" the ".NET Provider" for Firebird (recommended file "FirebirdSql.Data.FirebirdClient.dll "in version 2.5.2).

To use the class DataConnectionMySQL you must have previously installed on your system "MySQL Connector/Net" software, version 6.7.4 or higher (you can download http://dev.mysql.com/downloads/connector/net/).

At the head of the reference procedure be added to the class:

```
using SwanCSharp.DataAccess;
```

To create the connection with a Microsoft SQL Server database is executed:

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

In the first parameter is passed the name of the instance of SQL Server installed also indicating the name of the PC in Windows. The second parameter is the name of the database created in SQL Server, in the third parameter is passed an enumeration specifying that the database is Microsoft SQL Server. If Microsoft SQL Server is not installed locally, and is installed remotely via a LAN or WAN address accessible from the computer that will run the development, on the first instance will setting "IP ServerSQL \ \ INSTANCE SQL". But to make remote connections from SwanCSharp.DataAccess class, you must first have enabled remote connections to the Microsoft SQL Server installed on the remote computer, and given all necessary permissions. There is another overload that also allows passing the username and password for SQL authentication instead of Windows authentication.

To create the connection with an Oracle Server database is executed:

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

In the first parameter we pass the data source (for example in the case of Oracle 11g Express would "XE"). The second parameter is the SQL user, the third parameter is passed password. If what you want is a remote connection to Oracle, would include:

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

Being 192.168.0.17 the IP of the computer that has Oracle Server, 1521 the TCP port being communication (the standard port Oracle), and being XE data source in Oracle 11g Express installation.

To create the connection with a Firebird database is executed:

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

In the first parameter we pass the database with its complete path. The second parameter is the SQL user, the third parameter is passed password. If what you want is a remote connection to Firebird would use the second constructor of the class:

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

Being 192.168.0.17 the IP of Firebird server, and being 3050 TCP communication port (the Firebird standard port).

To create the connection with a MySQL database is executed:

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

The first parameter passed is the MySQL server address, the second parameter is the name of the database. The third parameter indicates the SQL user, the fourth parameter is passed user password.

There exists a second constructor of the class at which you can add the TCP communication port:

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

To create the connection with an Access database is executed:

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

In the first parameter is passed the filename Access (. Mdb) that contains the database. The second parameter is the full path location of the file on disk. The third parameter is the Access Password if you have one (assigned in Microsoft Access in Security -> Set Password), if no passphrase gets passed String.Empty.

After creating the connection, and regardless of whether of SQL Server, Oracle, Firebird, MySQL, or Access, there are two functions to execute commands, the first named SQLSelectExecute exclusive data for the database (using a standard SQL command), eg :

```
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());
}
```

A second command called SQLCommandExecute to implement the rest of standard SQL commands (INSERT, UPDATE, DELETE, etc.). For example to insert data into the database we use:

```
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");

 }
```

If we want to delete data from the database can be used:

```
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");
}
```

In the same way we can run other existing standard SQL commands such as UPDATE.

After creating a connection which executes all SQL statements needed, must ALWAYS close the connection, because if not closed will remain open in the data server and eventually crash. To close the connection created is used:

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

In the accompanying examples with this library (folder "Samples" coming within the ZIP file downloaded SwanCSharp.zip) an example of a Data Access project that operates against Access (the database is included) and against SQL Server (requires SQL Server installed and the database and table created).

Inside the classes "Connection" there are many predefined functions that can help us make small routine processes in databases without using source code to develop. Here we describe these functions.

10.4.1.1 Transactions

In all “DataAccess” class constructors (SQL Server, Oracle, Firebird, MySQL, and Access) exist the BeginTransaction, CommitTransaction, RollbackTransaction methods that allow us to manage data transactions.

Once you create a connection, for create a transaction, executes the method:

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

After running a set of SQL commands, to confirm the transaction, the method executed is:

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

If you want cancel the transaction use the method:

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

10.4.1.2 CalculateNextIntegerValue

This function allows us to calculate the next ID value to apply to a new record that is to be inserted. As the first parameter is passed the name of the table that we will insert the record, as a second parameter is passed the name of the field that has the characteristic of being the ID field. The function accesses this table and returns a value "Int32" with the next value that will ID the next row to be inserted.

In the header of the referenced method be added to the class:

```
using SwanCSharp.DataAccess;
```

An example of a call to the function is as follows:

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

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

10.4.1.3 CheckExistingDataInteger

This function checks if a chosen integer data exists in a given integer field within a specified table in the database.

In the header of the referenced method be added to the class:

```
using SwanCSharp.DataAccess;
```

An example of a call to the function is as follows:

```
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

This function checks if a chosen string data exists in a given string field within a specified table in the database.

In the header of the referenced method be added to the class:

```
using SwanCSharp.DataAccess;
```

An example of a call to the function is as follows:

```
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, and DatabaseStructureAccess

The purpose of the class is to dynamically create (at runtime) SQL Server, Oracle, Firebird, MySQL, or Access databases. This class is very useful for developing applications that can implicitly create and / or update the structure of their own databases for easy updating older versions as they develop new versions.

Can also be used to create proprietary applications that may create a database or update it.

In case of the "DatabaseStructureOracle" class DO NOT need to pre-create the database where data tables will be added, because Oracle does not work with databases but users's schemas. For this type of database data is ignored "Identity" in "DBSTRUCT" because there is no "AutoNumber" field type.

In case of the "DatabaseStructureFirebird" and "DatabaseStructureMySQL" class is necessary previous creation of the database where data tables will be created, because are not going to create it. For this type of database data is ignored "Identity" in "DBSTRUCT" because there is no "AutoNumber" field type.

In the case of "DatabaseStructureAccess" class you need to create an empty MDB file previously because "DatabaseStructureAccess" is not going to create it, simply use one empty file MDB to create the structure.

In the header of the referenced method be added to the class:

```
using SwanCSharp.DataAccess;
```

To create a data structure in SQL Server runs:

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

To create a data structure in Oracle runs:

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

To create a data structure in Firebird runs:

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

or

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

To create a data structure in MySQL runs:

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

or

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

To create a data structure in an Access file runs:

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

In both cases the last parameter, which we are passing "lobjStruct", refers to the existing structure in the class DBSTRUCT DataAccess. This structure consists of variables TableName, FieldName, FieldType, PrimaryKey, Identity, and NotNull. Creating an array of DBSTRUCT, we define the desired structure for our database. Then the DBSTRUCT array is passed to the constructor of the “DatabaseStructure” classes which is responsible for creating the entire structure.

Below is an example of creating a structure:

```
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);
```

In the previous example is defining a structure of two tables (Staff, Auxiliary) for DataBase. For the first table (Staff) defines two primary key fields and three normal fields. For the second table (Auxiliary) defines a primary key field and two standard fields. With this example we are creating dynamically from source code structure of a database in both Access and SQLServer.

When creating and defining the structure DBSTRUCT just have to consider some rules:

- Can simultaneously create all tables designed on one DBSTRUCT and in a single run of the DatabaseStructure case, but the fields in each table should be correlated within DBSTRUCT elements, i.e. can not intercalate fields of different tables.
- Each table must have at least a primary key field; otherwise the table will not be created and can have more than one primary key field.

- The "FieldType" is defined using the same types of data that exist in SQL Server and Access.
- The rules for creating the tables are the same as in SQL Server and Access, for example, you can not create a primary key field if it is a "false" variable in "NotNull" for the field, or you cannot assign the Identity property to a field type different of "int".

In the examples that come into the library ("Samples" folder that is inside the ZIP file downloaded SwanCSharp.zip) exist an example of a project "DatabaseCreation", operating against Access (the database is included) and against SQL Server (SQL Server is required to have installed and the database and table created).

10.4.3 DataExport

The class "DataExport" allows you to export an object "DataTable" to Excel XLS or CSV format, allowing export data from Access or SQL Server obtained by DataConnection class or any other DataTable loaded manually or obtained in another way.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.DataAccess;
```

A sample line of code can be:

```
// 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");
```

In the accompanying examples to the library ("Samples" folder that is inside the ZIP file downloaded SwanCSharp.zip) an example of a project "DataExport" that allows export data to Excel (CSV or XLS).

10.4.4 Utilities

The namespace "DataAccess" has a "Utilities" class which includes all functions that do not require a constructor to be called. These functions allow us to have some utility on databases.

10.4.4.1 DataGridViewToDataTable

The "DataGridViewToDataTable" function allows us to convert a control "DataGridView" (Windows.Forms) in a datatable.

At the head of the reference procedure be added to the class:

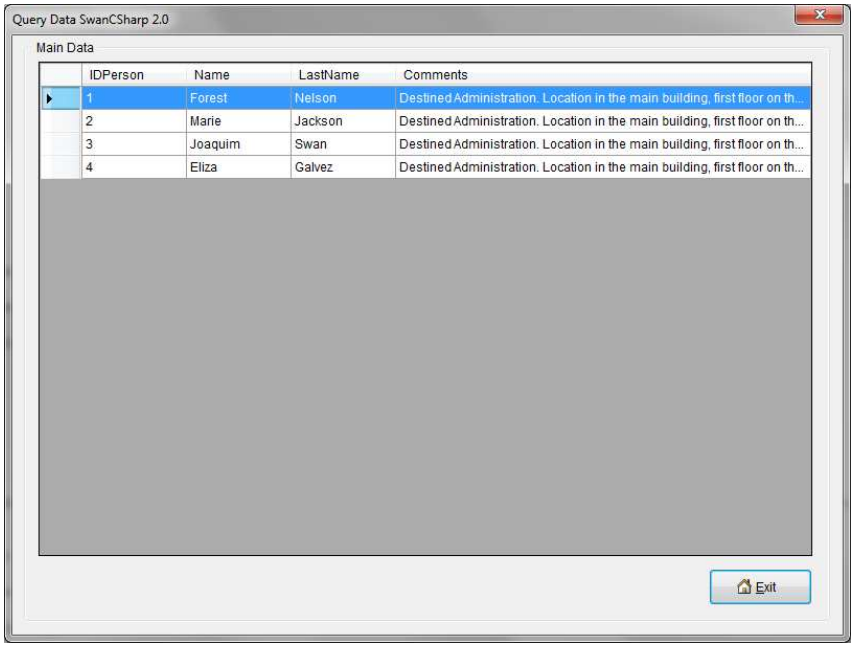
```
using SwanCSharp.DataAccess;
```

A sample line of code can be:

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

10.4.4.2 ShowQueryDataModifyWindow

The "ShowQueryDataModifyWindow" function allows us to display a window with a "Grid" data, modify the desired cells, and returns a "DataTable" with updated data:



The "ShowQueryDataModifyWindow" function expects to receive seven parameters which are: the DataTable with the data, an array of integers with datatable columns not want to show (if you want to display all, pass "null", and the number so each column is ordinal, with "0" in the first column), the desired title for the form to be displayed, the desired title for the "frame" shown on the form, the size of the window (using the listed QueryWindowSize choosing from Small, Medium, High), the language of the form (using the listed InterfaceLanguage, choosing between Spanish and English), and finally a "true" or "false" if you want the last column automatically expands to the full width of the grid, or not.

After running the function displays a Windows form showing the data on screen. The user can modify the data in the desired cells, clicking on "Exit" function will return a DataTable with all modified data.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.DataAccess;
```

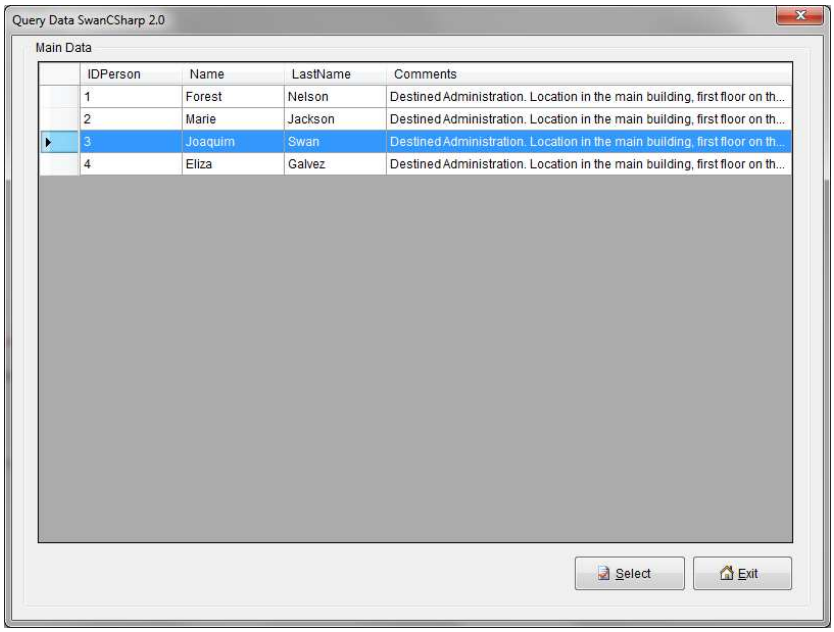
A sample line of code can be:

```
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 2.0", " Main Data ", QueryWindowSize.Medium,
InterfaceLanguage.English, true);
```

In the examples that come into the library ("Samples" folder that is inside the ZIP file downloaded SwanCSharp.zip) exist an example of a project "DataQueryAccess", operating against Access (database is included).

10.4.4.3 ShowQueryDataWindow

The "ShowQueryDataWindow" function allows us to display a window with a "grid" of data and returns a "DataRow" with the row that is selected in the window. A screen is as follows:



The "ShowQueryDataWindow" function expects to receive seven parameters which are: the DataTable with the data, an array of integers with columns that not want to show (if you want

to display all, pass "null", and the number so each column is ordinal, with "0" in the first column), the desired title for the form to be displayed, the desired title for the "frame" shown on the form, the size of the window (using the listed QueryWindowSize choosing from Small, Medium, High), the language of the form (using the listed InterfaceLanguage, choosing between Spanish and English), and finally a "true" or "false" if you want the last column automatically expands to the full width of the grid, or not.

After running the function displays a Windows form showing the data on screen. The user can select a line by double-clicking on the row, or click on the row and clicking the "Select" button. The function returns a "DataRow" with the data of the selected row.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.DataAccess;
```

A sample line of code can be:

```
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", " Main Data ", QueryWindowSize.Medium,
InterfaceLanguage.English, true);
```

In the examples that come into the library ("Samples" folder that is inside the ZIP file downloaded SwanCSharp.zip) exist an example of a project "DataQueryAccess", operating against Access (database is included).

Important note: In the case of custom forms "SwanCSharp" created from the namespace "SwanCSharp_Controls", you must use the class "WindowDataQuery" then existing in that namespace instead of this function "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

The "CopyDirectory" allows a source directory to copy in a different destination. The function takes two parameters <string>, the first with the source path, the second with the target path, and a third Boolean parameter which reports if you want the copy is recursive or not (besides recursive copy the folder and its files, copy all folders along with their dependent files from the original). The function returns a Boolean value reported if the process has been executed or not.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.5.2 GetFolderNamesRecursively

This function, given a path to a home folder, returns a string array with all existing folders in recursive tree structure.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.6 SwanCSharp.Encryption

The class "Encryption" incorporates a class to encrypt data and several features that allow encrypt and decrypt data in blocks of information. The cipher used is AES that can encrypt using MD5 or SHA1 HASH, using two keywords.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Encryption;
```

A sample line of code can be:

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

The first parameter of the constructor refers to the keyword that is used to encrypt a "string" with ASCII values freely chosen by the developer, the second parameter corresponds to the word to be used in combination with the first parameter to generate encryption (also a "string" free consists of ASCII values). The third parameter is supplied from existing HashFunction enumerated in the class and allows you to select whether you want to perform encryption using MD5 or SHA-1 (the second is more secure). The fourth parameter is used to report the number of iterations to generate desired encryption, with 2 iterations is enough. The fifth parameter refers to the initialization vector must always be a "string" of 16 ASCII characters long accurate. The sixth parameter refers to the class KeySize enumerated and you can select the length of the generated key, making it possible to choose between 128, 192, or 256 (if the key size is large, increase the security of encryption).

When data is encrypted with SwanCSharp.Encryption class, the six parameters used for encryption, shall be six parameters which must be used to decrypt the same string.

10.6.1 BytesEncryptToFile

It is used to encrypt a byte array and the result is saved directly in a specific file on a given path.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Encryption;
```

Then you build the class of encryption:
:

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

A sample line of code can be:

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

10.6.2 FileDecrypt

It is used to decrypt a file that was previously encrypted FileEncrypt function. Expects five parameters where the first four parameters collect information and the source file and destination file paths, and the fifth parameter is given a value "true" if you want to overwrite the destination file that may already exist, and false if you do not want to overwrite the file.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Encryption;
```

Then you build the class of encryption (user must report the same six parameters used to encrypt the file you want to decrypt):

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

A sample line of code can be:

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

10.6.3 FileDecryptToBytes

Used to decrypt an array of bytes in a file that was previously encrypted or BytesEncryptToFile FileEncrypt function. Expects two parameters with the source file name and path of the source file.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Encryption;
```

Then you build the class of encryption (user must report the same six parameters used to encrypt the file you want to decrypt):

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

A sample line of code can be:

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

10.6.4 FileEncrypt

It is used to encrypt a file. Expects five parameters where the first four parameters collect information and the source file and destination file paths, and the fifth parameter is given a value "true" if you want to overwrite the destination file that may already exist, and false if you do not want to overwrite the file.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Encryption;
```

Then you build the class of encryption (the user must choose the six parameters to use to encrypt the file, these same parameters should be used for decryption):

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

A sample line of code can be:

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

10.6.5 StringDecrypt

It is used to decrypt a "string" that was previously encrypted StringEncrypt function. Parameter is passed as the "string" to decrypt, and the function returns the second parameter a reference to the "string" decryption.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Encryption;
```

Then you build the class of encryption (user must report the same six parameters used to encrypt the string you want to decrypt):

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

A sample line of code can be:

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

10.6.6 StringEncrypt

It is used to encrypt a "string". Parameter is passed as the "string" to encrypt, and the function returns the "string" encrypted.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Encryption;
```

Then you build the class of encryption (the user must choose the six parameters to use to encrypt the file, these same parameters should be used for decryption):

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

A sample line of code can be:

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

10.7 SwanCSharp.Files

Files Class incorporates a number of support functions with file management and disk paths.

10.7.1 CheckPathEndsBackslash

The function receives a parameter <string> with a path of a disk, check if the path ends in backslash (\), if not already added it and returns the new <string>, if you already have the bar returns same string.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.7.2 CheckPathEndsSlash

The function receives a parameter <string> with a path of a disk, check if the path ends in slash (/), if not already added it and returns the new <string>, if you already have the bar returns same string.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.7.3 FileMove

The purpose of the function is to move a source file to a destination (indicating the name of the moved file). Expects three parameters and returns a bool value that reports whether it has moved or not. The first parameter is the file name to move with full path, the second parameter is the target file name with full path, in the third parameter is reported by a bool value if you want to overwrite the target file in case exist.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.7.4 FileToArrayBytes

The purpose of the function is to read a file and convert its contents into an array of bytes. Expects two parameters, the first with the path where the file is located, and the second with the filename. The function returns an array of bytes to the file contents.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.7.5 GZIPUncompressFile

The purpose of the function is decompressing a give GZip file in another target file passed by parameter.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.7.6 RemoveInvalidCharsFileName

The purpose of the function is removing from string all those characters that can not be part of a file name. The first parameter is file name, the second parameter must be defined by that character should replace those characters that are considered invalid. The function returns a string with the converted string.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.7.7 SaveArrayBytesToFile

The purpose of the function is to dump the contents of an array of bytes in a particular file in the desired location recording. The function takes four parameters, the first expects the array of bytes, and the second length of the array will be the size of the file when it is generated, in the third receives the path where to store the file, and in the fourth parameter is the name of the file to save. The function creates the file in the desired path, and returns true if the process has run successfully, and false if it failed to generate the file.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.7.8 SeparateFileNameAndPath

The purpose of the function is to receive a complete path of a file (path + filename) and separated into two different variables the path and filename. The first parameter is reported with the filename + path, and in the second and third parameter is received by reference a variable with path and other variable with the filename.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.7.9 UnZip

The purpose of the function is to decompress a ZIP file. The first parameter includes is the zip file to unzip with full path, and the second parameter contains the target path where you save the uncompressed files.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

In the examples that come to the library ("Samples" folder that is inside the ZIP file downloaded SwanCSharp.zip) exists an example of a ZIP decompression files.

10.7.10 Zip

The purpose of the function is compress into a single ZIP file one or more files located in a Path or in several different paths. The first parameter included zip file with full path (required include the path), and the second parameter contains an array of strings where each file is defined to be compressed. To generate the ZIP file is necessary for each file included in the array exists, and include the full path to the file.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

```
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);
```

In the examples that come to the library ("Samples" folder that is inside the ZIP file downloaded SwanCSharp.zip) exists an example of a ZIP compression files.

10.8 SwanCSharp.Imaging

Imaging class incorporates a number of functions to support the management of image files.

10.8.1 BitmapResize

The purpose of the function is to pick an image source, and from it create another image with the dimensions passed as parameter. The first parameter, expected to be received, is the source image, in the second parameter the width in pixels (integer), and the third parameter height in pixels (integer). The function returns a "Bitmap" with a new image with the specified dimensions.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.8.2 BitmapToMemoryStream

The purpose of the function is picking up a source image and convert to the data type MemoryStream. The first parameter expects to receive the source image, the second parameter it is expected to receive the copied image format. The function returns the object MemoryStream loaded.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.8.3 BitmapToUnsafeBytes

The purpose of the function is to pick up a source bitmap y and convert to Unsafe bytes (byte[]) data type. The first parameter contains the source bitmap. The function returns “byte unsafe” object. This function can be useful when you need to pass an image to a function created in C++.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.8.4 ByteArrayToBitmap

The purpose of the function is to pick up an array of bytes and convert to Bitmap data type, in the first parameter contains the array of bytes, and the function returns a "Bitmap".

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.8.5 ByteArrayToIcon

The purpose of the function is to pick up an array of bytes and convert to Icon data type, in the first parameter contains the array of bytes, and the function returns an "Icon".

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.8.6 ByteArrayToImage

The purpose of the function is to pick up an array of bytes and convert to Image data type, in the first parameter contains the array of bytes, and the function returns an "Image".

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.8.7 CompareImages

The purpose of the function is to compare two images passed by parameter returning true if they are identical and false if not.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.8.8 ConvertBase64ToByteArray

The purpose of the function is to collect a "String" Base64 format and dump on an array of bytes.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.8.9 ConvertBase64ToFile

The purpose of the function is to collect a "String" Base64 format and dump on a given file.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.8.10 ConvertFileToBase64

The purpose of the function is to collect a file and convert to the "String" Base64 data type format.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.8.11 ConvertBytesToBase64

The purpose of the function is to collect an array of bytes and convert to the "String" Base64 data type format.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.8.12 ConvertImageBytesToBase64HTML

The purpose of the function is to pick up an array of bytes of an image and convert it to Base64 format type to embed in HTML, allowing us to insert an image embedded in a web page using the tag "IMG SRC".

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

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

10.8.13 ConvertImageFileToBase64HTML

The purpose of the function is to pick up an image file and convert it to Base64 format type to embed in HTML, allowing us to insert an image embedded in a web page using the tag "IMG SRC".

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

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

10.8.14 ConvertTo8BppGrayscale

The purpose of the function is to pick an image file and convert it to grayscale with a depth of 8-bit (256 gray indexed).

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.8.15 DominantColor

The purpose of the function is picking up a source image and get the dominant colour of the image. In the first parameter receives the image, and the parameters two, three, and four are passed by reference a string for each RGB colour channel (red, green, blue). The load function in the last three parameters the dominant colour value broken down into channels.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

```
string lstrRed = "";
string lstrGreen = "";
```

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

10.8.16 GetDifferenceFromImages

The purpose of the function is to get the number of different pixels in two images that are passed as parameters.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.8.17 IconToByteArray

The purpose of the function is to pick up an icon object and convert to bytes array, in the first parameter contains the icon object, and the function returns an array of bytes.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.9 SwanCSharp.Internet

The class "Internet" allows developments incorporate utility functions related to the world of WANs and the Internet.

10.9.1 BreakdownFTPString

The purpose of this function is to collect (using string) string FTP connection such as "ftp://user:password @ip_ftp/AccessPath" and extract from it each element separately. The function returns a string array that contains the following values:

- Position 0. - Returns the user name to access the FTP.
- Position 1. - Returns the password to access the FTP.

- Position 2. - Returns the FTP server IP.
- Position 3. - Returns the FTP folder access.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Internet
```

A sample line of code can be:

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

10.9.2 CheckInternetConnection

The purpose of this function is to check for Internet connection on the computer.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Internet;
```

A sample line of code can be:

```
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, etc.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Internet;
```

A sample of code can be:

```
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

The purpose of this function is to obtain the domain name linked to a given valid IP.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Internet;
```

A sample line of code can be:

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

10.9.5 GetFileFromHttp

The purpose of this function is to download an uploaded file to the Internet via HTTP address.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Internet;
```

A sample line of code can be:

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

10.9.6 GetIPFromDomainName

The purpose of this function is to obtain the IP address linked to a given Internet domain name.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Internet;
```

A sample line of code can be:

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

10.9.7 GetWebProxyActive

The purpose of this function is to obtain all information from the web proxy that could be active on the execution computer, including credentials.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Internet;
```

A sample line of code can be:

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

10.9.8 IPToUint32

The purpose of this function is to convert a string standard IP address in a Uint32 data type.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Internet;
```

A sample line of code can be:

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

10.9.9 Uint32ToIP

The purpose of this function is to convert a Uint32 IP address into a standard string address.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Internet;
```

A sample line of code can be:

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

10.10SwanCSharp.Logger

The class "Logger" allows developments incorporate effective management of a system log and can publish in the log file error messages, warning, or information.

10.10.1 Logger

Order to use the Logger class must create the object passing two parameters: The first parameter is the path where you will generate each log file (if left blank, it will use the default path of execution), and the second parameter the name of our application.

When you run the constructor will create the folder "Log" in the path selected target, and into that folder will be created all log files.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Logger;
```

In order to create object "Logger" should add the line (in the developments in this class is used "Logger" is recommended to create this object globally) to publish messages from any class or function without having to new creation of this object:

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

Subsequently, each time you want to post a message, use the "WriteMessageToLog", allowing us to specify the message text, and the classification of the message (Error, Information, and Warning).

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

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

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

In the execution of each message, this function will create a file which will store the daily posts with the classification of the message, and the exact date-time. The names of the files created will begin with the application name passed in the second parameter in the constructor of this class.

10.11SwanCSharp.Miscellaneous

Miscellaneous Class is intended include all those functions and procedures that do not fit into any of the other major classes. We can define this class as a class that stores all generic functions.

10.11.1 CalculationDiskSpace

The purpose of the procedure is to calculate the free space available on a given drive (calculated in Bytes).

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.11.2 ComputerCloseSession

The purpose of the procedure is to close the open session in the operating system of the computer where the application runs.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.11.3 Delay

The purpose of the procedure is to produce an expected delay determined by the second parameter passed.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.11.4 ExistsLibrary

The purpose of the procedure is to check if a DLL or EXE file exists in the system. In the first parameter you enter the name of the library to search without specifying particular path on disk.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.11.5 FormCentering

This procedure is used to center a form created and screen visible. The parameter expects the form as an object.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.11.6 GetExecutionPath

This procedure is used to obtain the full path of execution of our application. You get clean path without file names, only folder structure.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.11.7 GetListOfCountries

This procedure is used to obtain the full list of countries (English or Spanish). When you run the procedure we will receive by reference receive two arrays of strings, one with the list of countries and the other with a numeric code for each country, so if you need to load it in a

normal SwanCSharp combo, or to load in custom graphic interface "SwanCSharp_Controls" combo-box.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

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

10.11.8 LoadDataInComboBox

This procedure allows to load all desired items in ComboBox, inserting each element with the “Text” and “Value” parameters desired (something that does not allow directly the ComboBox). This will create an array of string with the parameter "Text" for each item, and another array of "String" with the parameter "Value" of each item. The third parameter is passed by reference "ComboBox" we want to load.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

```
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());
}
```

If the ComboBox to load belongs to the class SwanCSharp.WindowBase, instead of using this function (SwanCSharp.Miscellaneous.LoadDataInComboBox), we must use the parallel SwanCSharp_Controls.Miscellaneous.LoadDataInComboBox function.

10.11.9 ObjectCentering

This procedure is used to center a control within its "parent object". The parameter expects the control as an object.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.11.10 RestartComputer

The purpose of the procedure is to restart the operating system on the computer where the application runs.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.11.11 ShowErrorMessage

The purpose of the procedure is to display a window with the error message, "OK" button, and the appropriate icon. As parameters pass the desired title and the message for the window.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

Important note: In the case of custom forms "SwanCSharp" created from the namespace "SwanCSharp_Controls", you must use the class "WindowError" then existing in that namespace instead of this function "ShowErrorMessage".

10.11.12 ShowInformationMessage

The purpose of the procedure is to display a window with the information message, "OK" button, and the appropriate icon. As parameters pass the desired title and the message for the window.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

Important note: In the case of custom forms "SwanCSharp" created from the namespace "SwanCSharp_Controls", you must use the class "WindowInformation" then existing in that namespace instead of this function "ShowInformationMessage".

10.11.13 ShowOKCancelQuestion

The purpose of the procedure is to show a window with a question and "OK" and "Cancel" buttons, and the corresponding icon. As parameters pass the desired title and message for the window for the window. Returns a "DialogResult" with the answer.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

Important note: In the case of custom forms "SwanCSharp" created from the namespace "SwanCSharp_Controls", you must use the class "WindowOKCancelQuestion" then existing in that namespace instead of this function "ShowOKCancelQuestion".

10.11.14 ShowWarningMessage

The purpose of the procedure is to display a window with the warning message, "OK" button, and the appropriate icon. As parameters pass the desired title and the message for the window.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

Important note: In the case of custom forms "SwanCSharp" created from the namespace "SwanCSharp_Controls", you must use the class "WindowWarning" then existing in that namespace instead of this function "ShowWarningMessage".

10.11.15 ShowYesNoQuestion

The purpose of the procedure is to show a window with a question and "Yes" and "No" buttons, and the corresponding icon. As parameters pass the desired title and message for the window for the window. Returns a "DialogResult" with the answer.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

Important note: In the case of custom forms "SwanCSharp" created from the namespace "SwanCSharp_Controls", you must use the class "WindowYesNoQuestion" then existing in that namespace instead of this function "ShowYesNoQuestion".

10.11.16 ShutDownComputer

The purpose of the procedure is to shut down the computer where the application runs.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.11.17 TextSlicingInLines

The purpose of this function is to slicing received a long text in a "string" in the desired line width. This function can be embedded long texts of paragraph in desired lines. To perform

the cutting of each line is calculated for each word length to avoid cutting through the middle, passing the entire word to the next line.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.12 SwanCSharp.Network

The "SwanCSharp.Network" class allows us, using methods and functions, manage some aspects of a computer network.

10.12.1 CheckITCPPortIsOpen

In the "Network" class exists "CheckTCPPortIsOpen" function which tells us whether a TCP port of a given address or domain name is open or not.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

10.12.2 GetIPNetworkData

In the "Network" class exists "GetIPNetworkData" function that returns all the information associated with the Ethernet card system. The function returns an array of object "NetworkData" containing the caption, description, MACAddress, DHCPEnabled, DHCPServer, DnsDomain, DNSHostName, IPAddress, IPSubnet, DefaultIPGateway, DNSServer and SettingID fields.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

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

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

10.12.3 IsLocalIP

In the "Network" class exists "IsLocalIP" function that tells us whether a given IP is local or not.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

```
Boolean lblnIPLocal = Network.IsLocalIP("255.255.255.255");
```

10.12.4 IsValidIP

In the "Network" class exists "IsValidIP" function that tells us whether a given IP address has the correct structure and is valid.

At the head of the reference procedure be added to the class:

```
using SwanCSharp;
```

A sample line of code can be:

```
Boolean lblnIPValid = Network.IsValidIP("255.255.255.255");
```

10.13 SwanCSharp.Reporting

The namespace "SwanCSharp.Reporting" allows us to generate data reports for later viewing and / or printing. This class incorporates a viewing window itself from which to print the report.

Was decide to use HTML files for reports by its universality, can be viewed on any system, because of the small space they occupy, and flexibility.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Reporting;
```

A sample line of code can be:

```
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);
```

The above code will generate a report using data from ldatData, and the definition of the object has passed "ReportObjects". The report will be saved in the specified path. The reporting parameters are:

* DataTable. - The data used to generate the report.
* ReportObjects. - The object constructed with the basic structure of the report that will allow its construction.
* Filename. - The name of the file you will when stored on disk. The file name and extension will be passed without SwanCSharp will add the extension. "Html".
* File path. - The disk path where it will create the html file.
* Report title. - The title is going to have the report.
* Filename logo. - You can pass the name of the image file to be used as a logo to include in the report.
* Logo file path. - The disk path where the image file of the logo.
* Interface language screen. - We choose the language in which to display the report output.
* Size of the display window. - You can choose between Small, Medium, High.
* Date of creation of report. - If you pass "false" in this parameter will be printed at the end of the report the date and time of creation.
* Viewing the report. - If you pass "false" in this parameter, once generated in the report will show a preview screen automatically.

The report generator was created with the idea of displaying data whose origin is a "DataTable" that I could get from any external form, or using the class of SwanCSharp DataAcess (as seen in the example above). Although the report will in many sections show a text "constant". The idea of "Reporting" is to generate reports specific facts of a DataTable.

To create a report is essential to use the class "ReportObjects" which is the class that will allow us to customize the generation, allowing constant data entry, set a particular text in bold or underline, set alignment of text or information, place a title bar, or place the detail where to display the data.

The first step in designing a report with "Reporting" is split in specific rows report, for example, a standard report can be divided into three general lines, a header row, where general data will show a detail row where to display more specific data, and a row at the bottom of the document output. In this document we show an example of how to create a report that will have a front row with more general data and a second row with the detail of the specific information.

To create each particular row can use three different classes: ReportRowStandard, to create a single row with general data (the data you wish), that equals ReportRowDoubleBox above only allows us to chop the line into two distinct parts and letting you configure the width of each of the two parts, and finally ReportRowDetail, which is used to create a wider detail.

Let's show a particular example. We want to create a report for each record in your database (Table staff) we show a first row divided into two parts, the first to show only the code of the employee, and the second to show the rest of personal data. Then there will be a second row with the detail of the other fields.
For the first row we choose an object "ReportRowDoubleBox". We create the object as follows.

```
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;
```

In the above example we can see how two objects "ReportRowData" (which is the basic object of Reporting), one for the first part of the row, and one for the second part of the line. In each part we can create an array with many elements want to display data in our case shows two data for the first half and two data for the second part (may be different in each part).

Once created the two parties are going to merge to create the entire row by ReportRowDoubleBox object:

```
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;
```

In ReportRowDoubleBox object can be seen as we pass the object of each party, and we can move separately for each part if you show the border, or text alignment. This object is also determines the width that will be the first part of the box in this case is 30, which means that the width of the first part is 30%, therefore the width of the second part will be 70% automatically.

When this object is passed to the generator, which will be created in the report will be:

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

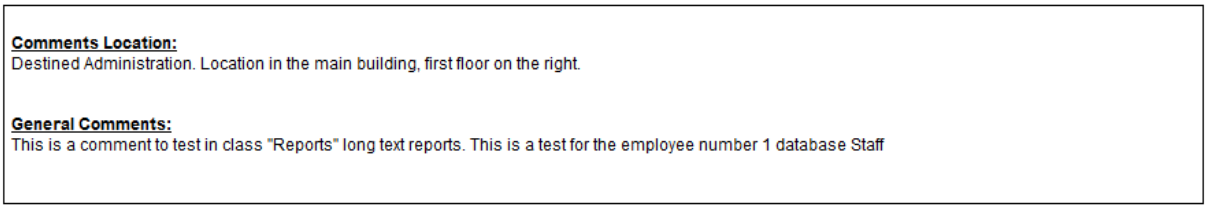
The next step is to create the detail row, and this class is used as base ReportRowDetail using each data the same class "ReportRowData" used in the previous case:

```
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;
```

As you can see in the above code we will show two fields in the detail; now create the ReportRowDetail object:

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

For this purpose we have chosen detail portrait orientation, text alignment left and show the border. When this object is passed to the generator, which will be created in the report will be:



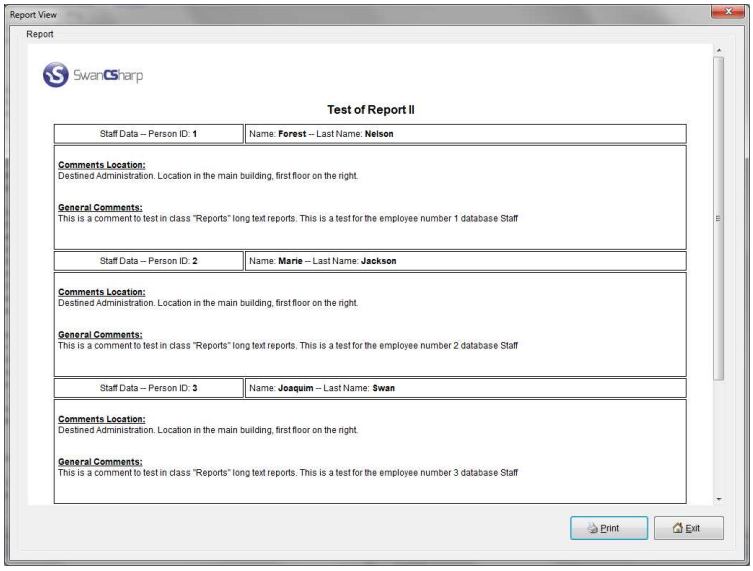
In short, from the base class ReportRowData have created a report which is divided into a first ReportRowDoubleBox object and a second object ReportRowDetail.
But to generate the report must pass a ReportObjects. The next step is to create ReportObjects from each of the two objects "Row" (you can check that we can create as many objects within ReportObjects row as we wish):

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

The last step is to call, passing the object ReportObjects, the class that will create the HTML file to view the report. This file can be viewed in any web browser:

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

The class "GenerateHTMLReport" create report "Report.html" and display it on screen automatically, showing the following window:



In the examples that come with the lib ("Samples" folder that is inside SwanCSharp.zip downloaded ZIP file), exists a project with examples of "Reporting" that is named "Report".

10.13.1 ReportHTMLViewer

ReportHTMLViewer class allows us to show in a window any existing HTML report without rebuilding it. It is necessary to report the following parameters:

* Filename. - The name of the file you will when stored on disk.
* File path. - The disk path of the file.
* Title window. - The title to be displayed in the window.
* Group title. - The title to display in control group.
* Size of the display window. - You can choose between Small, Medium, High.
* Interface language screen. - We choose the language in which to display the report output.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Reporting;
```

A sample line of code can be:

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

Important note: In the case of custom forms "SwanCSharp" created from the namespace "SwanCSharp_Controls", you must use the class "WindowReportView" then existing in that namespace instead of this function "ReportHTMLViewer".

10.14SwanCSharp.SNTP

The class "SwanCSharp.SNTP" allows us to integrate our applications an upgrader date and system time. There are many free time servers (NTP Server, Network Time Protocol Server) that allow us to get a date and time updated and certified. With a few simple lines of code we can make our applications synchronize (or update) the date and time by connecting to a NTP server.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.SNTP;
```

A sample line of code can be:

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

The class constructor expects the DNS or IP address of the NTP server that we will connect. The method "Connect" expects to receive a parameter with time out interval (in seconds). By running these two lines of code our application will connect to the NTP server, pick the current date and time, and modify the system clock with the new date and time.

10.15SwanCSharp.Socket

The class Socket incorporates all the features needed to manage sockets in applications developed in. NET Framework. The class is divided into four modules, one to run as a server to receive all communications, and a second module as a client to send all commands to the server. Then there is a third module to enable a files server and a fourth module to enable a client files.

The class allows to manage develop systems Sockets File Transfer, Chat and communications applications, send / receive images via TCP, database applications using client / server communication client machine to server machine to perform configuration changes one core of a process computer, etc., can develop thousands of useful applications easily.

For example, you can develop an application to download files hosted on a server with requests from multiple clients, without requiring a server and an FTP client or SSH installed on each machine, without relying on a defined command structure, and can customize all commands.

Can also be used to house a data recording engine is supported on a central DB to server mode and record and request information from each customer, this prevents us develop a website that does not have the flexibility of an application Windows forms.

The communication between client and server is encrypted SwanCSharp socket for safety and operates by transmitting the whole "Command" and "Data" so the server socket can not

be used with another client socket stranger to this library, and the client can not be used in communication with a server other than the socket of this library.

This class can be used both Socket communication between client and server via LAN and via WAN (Internet).

10.15.1 FileClient

FileClient class lets you create a client socket to send or receive files to FileServer class.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Sockets;
```

To create the client object be used the commands:

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

To upload or download files from the server are used:

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

After sending the message, the event `TransmissionResultEventHandler` jumps when a server response to our command and the requested transfer. The event we have signed in the second line of the client connection, so we need to manage the event by adding the function:

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

Within the previous function we receive the file name in the transaction, if the transaction is "Download" (true) or "Upload" (false), and the result of the transaction (true correct false error). For example:

```
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");
   }
}
```

In the examples that come to the library ("Samples" folder that is inside the ZIP file downloaded SwanCSharp.zip) exists an example of a client project files that operates in combination sample project file server.

10.15.2 FileServer

FileServer class lets you create a server socket on a computer to be used to store files of any type (file server). The file server expects to receive orders from file client to upload or download a given file. FileServer class should run against one or more client computers that operate using FileClient class, so there is a direct relationship between classes FileClient and FileServer. Both classes allow us to transfer files without having to operate using standard protocols FTP, SSH, etc. FileServer and FileClient classes may be combined with SocketServer and SocketClient classes, to send a TCP channel to execute the commands, and a second channel to transfer files, so that the slower communication files will not interfere with communication faster commands and data. The maximum size of each file to be sent should not exceed 20 MB.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Sockets;
```

To create the server object and set it to listen commands, are used:

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

It is reported as the first parameter is the IP that the computer on which to run the server, passing as the second parameter desired TCP port for communication between client and server. The third parameter is used to specify if a file to be downloaded must be deleted from the server. After running the above lines, the application will remain in listening TCP port chosen.

When a client connects to the file server, run the event "OpenClientConnectionEventHandler" therefore we declare the event and PROCEDURE where we will process:

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

Procedure which is to be processed:

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

Each time a client sends data through the event "ReceiveDataEventHandler" will receive five parameters: Process File, Client IP, TCP Client, a Boolean value indicating whether the file is downloaded (true) or if the file is uploaded (false), and the communication channel).

To subscribe to the event runs:

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

To manage the client received frame use:

```
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());
     }
}
```

In the examples that come to the library ("Samples" folder that is inside SwanCSharp.zip downloaded ZIP file) there is a project management developed example of a file server.

With the simple lines of code described in this section have a file server that transmits operational information maximum speed advantage that allows us communication, either on a local network LAN and WAN.

10.15.3 SocketClient

SocketClient class lets you create a client socket to send commands, data, and information to the class SocketServer.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Sockets;
```

To create the client object are used the commands:

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

To send a command + data to the server is used:

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

To close the connection is used:

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

To send a command and data connection is recommended to create the client, send the command and data, receive the reply, and then close the connection. Socket Server supports multiple clients connected, but not convenient to use a single open connection to send multiple commands, you should create the connection and send the command to close each of the items.

After sending the message, the event `ReceiveDataEventHandler` starts when a server response to our command and data sent. The event we have signed in the second line of the client connection, so we need to manage the event by adding the function:

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

Within the above function to pick up the answer to Server command sent along with the data you send us, and make the processes that may be required. For example:

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

```
}
```

In the examples that come to the library ("Samples" folder that is inside the ZIP file downloaded SwanCSharp.zip) an example of a client project that operates in combination Socket project SocketServer example.

10.15.4 SocketServer

The SocketServer class lets you create a socket server to receive commands, data, and information SocketClient class.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Sockets;
```

To create the server object and set it to listen commands, are used:

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

Passing as the first parameter is the IP that the computer on which to run the server, passing as the second parameter the desired TCP port for communication between client and server. After running the above lines, the application will remain in listening TCP port chosen.

When a client socket connects to the server, execute the event "OpenClientConnectionEventHandler" therefore we declare the event and procedure where we will process:

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

Procedure which is to be processed:

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

Each time a client sends data through the event "ReceiveDataEventHandler" will be received five parameters: Command, Data, Client IP, TCP Client, and the communication channel).

The command is a two-character string that matches a hexadecimal value ranging from 00 to FF, allowing 256 receiving different commands. The second parameter is receiving the data (a string with no limit). The third and fourth parameters receive the client IP and TCP port

information we received. The last parameter is used to send a confirmation signal or customer error.

To subscribe to the event executes:

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

To manage the frame client received use:

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

In the above function, when you want to return a message to the client to indicate that the command is right or wrong, for example, can send a command "FF" when it's right, and a command "F0" when it is wrong, for that we use the pobjChannel parameter received:

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

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

In the examples of this library ("Samples" folder that is inside SwanCSharp.zip downloaded ZIP file) commands operate with "01", "02" and "03" for three different processes, and use the command "FF" to confirm success and "F0" to confirm error. But the number of commands to use, and which commands you choose is decision of each developer, should always be a two-character string that matches a valid hexadecimal number, but each developer can choose which you want.

With the simple lines of code described in this section will have a socket server running and encrypted communication, certifying that the recipient gets properly controlled by the message CRC. Depending on the imagination can develop amazing apps with this socket.

10.16SwanCSharp.Users

The namespace "Users" allows us to manage everything related to users and profiles. With the classes defined in this namespace can integrate into your applications with few lines of code, and easily, complete management of users.

10.16.1 UserManagementSQLServer, UserManagementOracle, UserManagementFirebird, UserManagementMySQL y UserManagementAccess

The class constructor "UserManagement" need to use a specific data table either SQL Server, Oracle, Firebird, MySQL, or Access (one class for each database manager). The purpose of this class is to incorporate an application complete user management (including management windows). So every time you build the object "UserManagement", this class will check our database tables if exist "Profiles" and "Users" and if they have the required fields. Otherwise the first thing it will automatically create all the necessary structure in our database chosen (a process that only runs the first time and without developer intervention).

In the case that the chosen data source is SQL Server, the data from the table "Users" will be encrypted for class "Management" so that no externally to our developments can access user data and passwords. Therefore, this class uses internally security encryption.

In the case that the chosen database is Access, the data is not encrypted, but users are recommended to protect the database with a password (from the Microsoft Access Security option).

Data is not encrypted for Oracle, MySQL, and Firebird, therefore we recommend that you enable SQL authentication that allow both database managers.

At the head of the reference procedure be added to the class:

```
using SwanCSharp.Users;
```

When building the UserManagement class object must remember that in the database passed are created the tables "Users" and "Profiles". The profiles will be created are: super user, administrator and standard user. With these three profiles can play in our developments to set permissions.

Regarding the table "Users" always creates a default user "superuser" profile, with which we can begin to create our desired users. The user base is created is "**superadmin**" and password is "**sadmin**".

It is also important to highlight that the classes "UserManagement" will be used throughout the development of our application and therefore it is recommended to create the object using a global variable so that the current user's data, and calls to management functions , are accessible from anywhere in the application. To create object "UserManagement" be added:

```
public UserManagementSQLServer gobjUserManage = null;

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

```
}
```

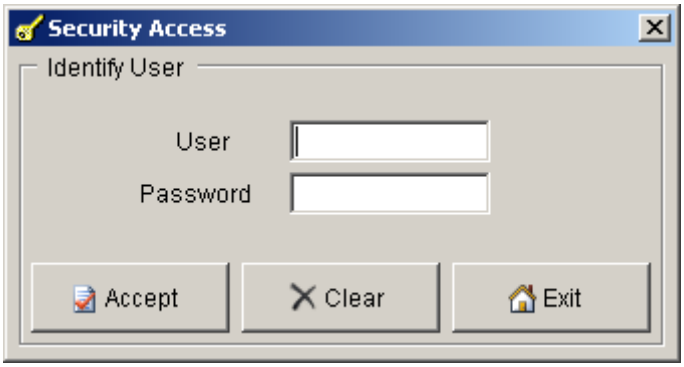
To create the object from Oracle, Firebird, MySQL, or Access, exists "UserManagementOracle", "UserManagementFirebird", “UserManagementMySQL”, and "UserManagementAccess" classes.

10.16.1.1 ShowLoginWindow

The logical thing would when running the application will display a window "login" to verify a correct user entry. So before opening main form "Form1" we call the window "Login". Then the source code would look like this:

```
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();
   }
}
```

Windows forms of this class can be displayed in two languages through enumerated InterfaceLanguage, Spanish and English. The above code will display a login window, if the user is not correct, the application will close. From the creation of the object "User Management" have a global variable "gobjUserManage" available throughout the application that has properties with the values of the user who accessed via login (UserName, UserPassword, UserCompleteName, Profile).



Once we have a public variable in order "UserManagement" can control, for example, options on a C # MenuStrip are enabled by the active user profile. For example, by the following source code:

```
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;
    }
}
```

If the active user "logged in" has the profile of "Super User", has access to all menu options. If the user has the profile "administrator" will not have access to the "Delete" and to the "Add User". If the user has a profile "StandardUser" can not access the menus "Save", "Setup" and "Delete". The "MenuManage" runs just after confirming that access login is correct and the permissions will be set as your profile menu.

Important note: In the case of custom forms "SwanCSharp" created from the namespace "SwanCSharp_Controls", you must use the class "WindowLoginUser" then existing in that namespace instead of this function "ShowLoginWindow".

10.16.1.2 ShowUserAddWindow

In addition to the "login" window exists "Add User" window will allow us to add an option to register new users. An example of the window called "Add User" is:

```
private void mnuAddUser_Click(object sender, EventArgs e)
{
   gobjUserManage.ShowUserAddWindow(gobjUserManage.Profile,
InterfaceLanguage.English);
}
```

As the first parameter is passed the current user's profile at every instant, and the second parameter is passed the language of the user interface. An example screen is:

This screen is never going to allow a user profile "StandarUser" create another user. A profiled user "Administrator" only allows you to create another user "Administrator" or another user "StandarUser". For the user "SuperAdmin" no restrictions whatsoever.

Important note: In the case of custom forms "SwanCSharp" created from the namespace "SwanCSharp_Controls", you must use the class "WindowAddUser" then existing in that namespace instead of this function "ShowUserAddWindow".

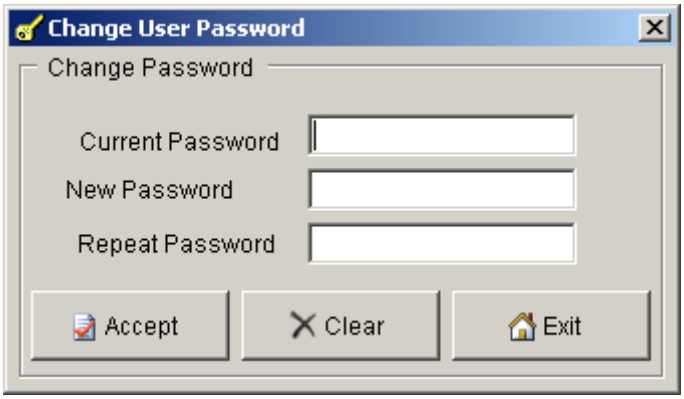
10.16.1.3 ShowUserPasswordChangeWindow

The window "Password Change" will allow us to change the password of the current user that is logged in at the time. This method opens a window where you will enter the current password, and will enter the new password. Pressing "OK" the password will be changed as long as the current password entered matches.

To call the password change window write the following code:

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

As the first parameter is passed the active user at every moment, and the second parameter is passed as the language of the user interface. An example screen is:



Important note: In the case of custom forms "SwanCSharp" created from the namespace "SwanCSharp_Controls", you must use the class "WindowPasswordUser" then existing in that namespace instead of this function "ShowUserPasswordChangeWindow".

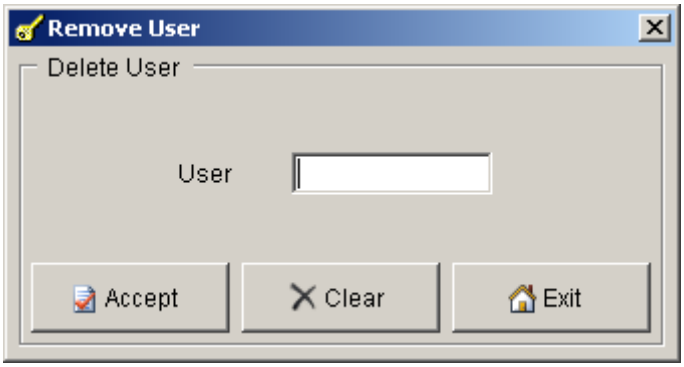
10.16.1.4 ShowUserRemoveWindow

The window "Remove User" will allow us to remove users from the database. To remove a user only need to enter his username. It is important to know that there are some rules to delete users: If the current user is "SuperUser" can delete any other user whatever their profile, if the current user's profile is "Administrator", can delete only "Standard User". The "Standard User" does not have permission to delete any other user whatever their profile.

To invoke the user deleted window you write the following code:

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

As the first parameter is passed the current user's profile at every instant, and the second parameter is passed as the language of the user interface. An example screen is:



Important note: In the case of custom forms "SwanCSharp" created from the namespace "SwanCSharp_Controls", you must use the class "WindowRemoveUser" then existing in that namespace instead of this function "ShowUserRemoveWindow".

10.16.1.5 ChangeUserPassword

With "ChangeUserPassword" function we can change the password of an active user from source code.

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

10.16.1.6 IsValidUser

With "IsValidUser" we can, from source code (the login screen and automatically used), to check if a given user is valid. This function is used to check if a username and password exist in the users table. In addition to returning "true" if exists, and "false" if it does not exist, returns by reference the user's full name and the profile of the user (if exists).

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

10.16.1.7 ExistsUserName

Given a username, the “ExistsUserName” function checks if the username exists in the table "Users". This check is done automatically in the window "Add User", but by source code we may want to check if a user exists or not.

```
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

The "ProfileUserName" allows us to obtain a user's profile according to the parameter passed.

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

10.16.1.9 UserAdd

The "useradd" function lets you create a new user via source code, without using the user creation window.

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

10.16.1.10 UserRemove

The "UserRemove" function allows us to remove a user determined by parameter. For the deletion is executed will have to meet deletion standards specified in this document under "UserManagement".

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

10.17SwanCSharp.Validations

It incorporates all the functions needed for rapid validation of data in applications developed in. NET Framework.

10.17.1 FileNameWithPathValidate

The purpose of this function is to validate if a string value passed parameter is a valid full path of a file. The function performs the following validations:

- Verify that the file exists. Otherwise it returns false.
- Check that the definition of the file name includes the path. Otherwise it returns false.
- Extract filename and checks the path exists. Otherwise it returns false.

In the header of the referenced method be added the class:

```
using SwanCSharp;
```

At the desired location to execute the validation be added:

```
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

The purpose of this function is to validate whether a string is a valid hexadecimal string.

In the header of the referenced method be added the class:

```
using SwanCSharp;
```

At the desired location to execute the validation be added:

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

10.17.3 HigherNumber

The purpose of this function is to return the greatest number of two numbers passed as parameter.

In the header of the referenced method be added the class:

```
using SwanCSharp;
```

At the desired location to execute the validation be added:

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

10.17.4 IsNumeric

The purpose of this function is to check if a string (or char) is given a numerical value or not. The function has four overloads to return from a single parameter to three parameters and can pass the number as a string, a char, and or can specify what the decimal point or the number of decimal places.

In the header of the referenced method be added the class:

```
using SwanCSharp;
```

At the desired location to execute the validation be added:

```
string lstrNumber = “215.67”;
Boolean lblnIsNumeric =  Validations.IsNumeric(lstrNumber, “.”, 2)
```

10.17.5 IsValidDate

The purpose of this function is to check whether a received date (in data type "string") is a valid date in calendar and format.

In the header of the referenced method be added the class:

```
using SwanCSharp;
```

A sample code can be:

```
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

The purpose of this function is to check if a given email has the correct format for an email address.

In the header of the referenced method be added the class:

```
using SwanCSharp;
```

A sample code can be:

```
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

The purpose of this function is to return the smallest number of two numbers passed as parameter.

In the header of the referenced method be added the class:

```
using SwanCSharp;
```

At the desired location to execute the validation be added:

```
int lintLowerNumber =  Validations.LowerNumber(10,15);
```

10.18SwanCSharp.Video

The "Video" namespace allows us to easily obtain one video stream generated by an IP camera with HTTP support. The IP cameras that can get video stream via http.

In the header of the referenced method be added the class:

```
using SwanCSharp.Video;
```

The "Video" namespace contains a main class called “Streaming” with two constructors: one only need to specify the http address of the CGI which returns every JPEG; the second addition to the http address allows us to customize the interval (in milliseconds) with which we want to download each image frame.

For example, if we pass only the http address, each frame is downloaded every 10 milliseconds, and the constructor would be:

```
Streaming mobjStreaming = new Streaming("http://192.168.x.x /axis-
cgi/jpg/image.cgi");
```

In this case we are setting the http address for downloading each frame of AXIS camera; each camera manufacturer will have its own http address.

If we want to customize the download time of each frame, for example to download a frame every 100 milliseconds (10 fps), would be:

```
Streaming mobjStreaming = new Streaming("http://192.168.x.x /axis-
cgi/jpg/image.cgi", 100);
```

If the camera has set username and password authentication, add the following:

```
mobjStreaming.User = "user";
mobjStreaming.Password = "password";
```

The last step is to set the event will jump with each new frame, and start capturing frames:

```
mobjStreaming.NewFrame += new
Streaming.NewFrameEventHandler(mobjStreaming_NewFrame);

mobjStreaming.StartCapture();
```

We can only create the event method "newFrame" which jumps to each gathered frame arriving by parameter "Bitmap" object:

```
private void mobjStreaming_NewFrame(Bitmap pobjFotograma)
{
      /*
       * Code for "pobjFotograma" image management here
       */
}
```

If at any time you want to stop capturing frames, you can use the following command:

```
mobjStreaming.StopCapture();
```

11. FUNCTIONS USER REFERENCE (SwanCSharp_Controls)

Here will describe the reference of use of each of the features incorporated into the library in SwanCSharp_Controls namespace. References are grouped by the classes to which they belong in the logical order of development.

11.1 SwanCSharp_Controls.WindowBase

The class "WindowBase" consists of an heritable object that allows us, in our developments, breaking the windows GUI that provides the operating system, and operate with another different interface, extending the limited customization options in a standard development.

For example, in a standard development you can hide the button separately to minimize and maximize, or we can disable all buttons (ControlBox) at once, but we can not hide only close button, and if we hide all (ControlBox), not may include a graphical icon in the title bar of the window. With SwanCSharp WindowBase class can hide each button separately, and includes the icon to the window anyway, we even can change the order of buttons close, maximize / restore, and minimize.

We can also change the colour text of the title bar, something that can not be done on standard interface and can block the possibility of move the window (Moveable) characteristic removed since Visual Basic 6.0.

The WindowBase class enables the possibility to include background transparency for windows and a lot of functions that are passed following paragraphs describe.

The "SwanCSharp_Controls" graphical interface includes six different themes to choose; each item is based on a specific central color (black, blue, gold, gray, green, and red).

To use the namespace "**SwanCSharp_Controls**" must be installed on the computer version 3.0 of .Net Framework or higher.

Respect to the development IDE, can be used without problems on Visual Studio 2008 or higher. Respect to Visual Studio 2005 will have to install the module "**Visual Studio 2005 extensions for. NET Framework 3.0 (WPF)**", which can be downloaded free from Internet.

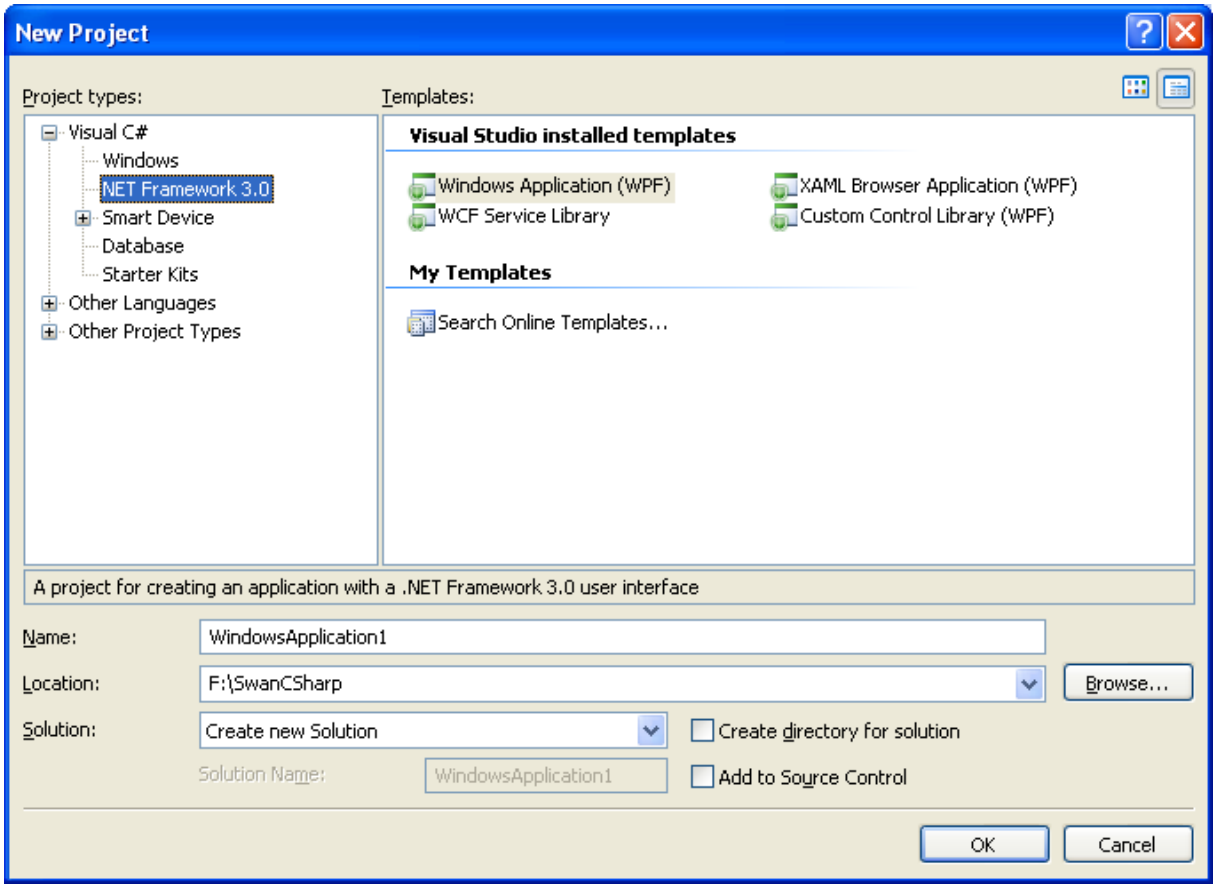
Without the installation of this module can not be developed using the namespace SwanCSharp _Controls.

The examples shown in this document are executed under the Visual Studio 2005 with "Visual Studio 2005 extensions for. NET Framework 3.0 (WPF)" installed.

To display general operation of the WindowBase class we created a sample project called "SwanWindow" that shows most important graphical features of this class. Running the sample application will ask for a username and password, you must enter "admin" in both cases.

11.1.1 New project creation

To create a new project you can work with the new style of windows and controls need to go to the menu File -> New Project. In the window displayed on the screen must be selected (left) under "Visual C #" a project. "NET Framework 3.0" and later (right) "Windows Application (WPF)". You select the desired name to the project and folder where you will store and click on the button "OK".



11.1.2 Creating a "SwanCSharp" window

When you create a new WPF project will create an **App.xaml** initial file that will run a initial window **Window1.xaml** call. In order to create a "SwanCSharp" window it is necessary to reference the project file "SwanCSharp.dll". You also need to reference the .Net Framework class project "System.Drawing".

After SwanCSharp reference, change the Window1.xaml file because standard class inherits from Window, and we want that file inherits from WindowBase class exists in SwanCSharp_Controls. Inheritance is changed as follows:

- We eliminate from Window1.xaml labels `<Grid></Grid>`.
- In Window1.xaml we change the label name `<Window x:Class="WindowsApplication1.Window1"` by `<src:WindowBase x:Class="Pruebas2.Window1"`.
- In Window1.xaml we change the label name `</Window>` by `</src:WindowBase>`.
- Below the line `xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"` we add the new line with WindowBase class reference that is `xmlns:src="clr-namespace:SwanCSharp_Controls;assembly=SwanCSharp"`.
- In Window1.xaml.cs file we add the reference to the namespace line SwanCSharp including "`using SwanCSharp_Controls;`".
- In the file Window1.xaml.cs we change the line "`public partial class Window1 : System.Windows.Window`" by "`public partial class Window1 : WindowBase`". In this way we tell that're going to inherit from WindowBase.
- In the main constructor of the class (`public Window1()`) below you have to add "`base(true)`", leaving the constructor "`public Window1() : base(true)`". The "`true`" value enables background transparency to our window if we do not want this transparency change it to "`false`". In this overload of the constructor, the central colour interface (theme) will always be blue.
- There is a second main constructor overload of the class (`public Window1()`) that lets you select the center color of the application (subject) besides selecting the transparency on/off. For example: "`base(WindowTheme.Black, false)`". There are the following themes: `WindowTheme.Black`, `WindowTheme.Blue`, `WindowTheme.Gold`, `WindowTheme.Gray`, `WindowTheme.Green`, `WindowTheme.Red`.

Once we get changes, from the original file 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>
```

We obtain the modified file 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>
```

From Window1.xaml.cs original file:

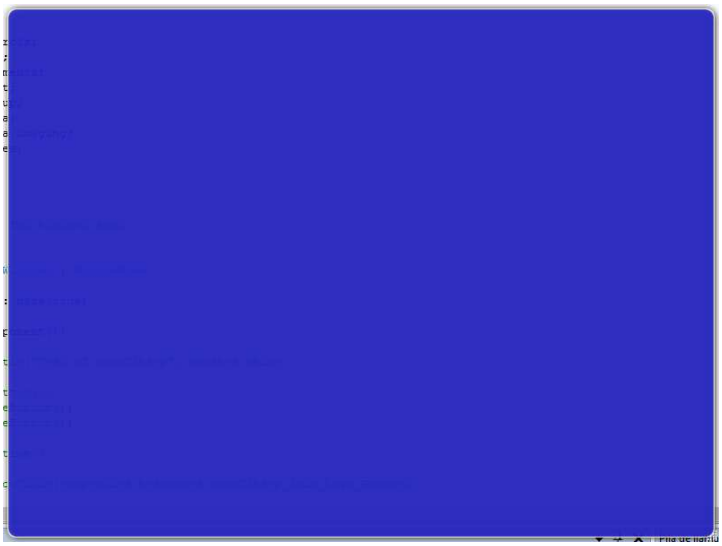
```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;


namespace WindowsApplication1
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>

    public partial class Window1 : System.Windows.Window
    {

        public Window1()
        {
            InitializeComponent();
        }

    }
}
```

We obtain the modified file 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();
        }

    }
}
```

In executing the project WindowsApplication1 we will display the window style "SwanCSharp" including transparency on the background, and several advantages that will be explained throughout this document.



For `WindowTheme.Black` theme:

```
namespace WindowsApplication1
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>

    public partial class Window1 : WindowBase
    {

        public Window1() : base(WindowTheme.Black, true)
        {
            InitializeComponent();
        }
```

```
    }
}
```

For `WindowTheme.Gold` theme:

```
namespace WindowsApplication1
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>

    public partial class Window1 : WindowBase
    {

        public Window1() : base(WindowTheme.Gold, true)
        {
            InitializeComponent();
        }

    }
}
```

For `WindowTheme.Gray` theme:

```
namespace WindowsApplication1
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>

    public partial class Window1 : WindowBase
    {

        public Window1() : base(WindowTheme.Gray, true)
        {
            InitializeComponent();
        }

    }
}
```

For `WindowTheme.Green` theme:

```
namespace WindowsApplication1
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>

    public partial class Window1 : WindowBase
    {

        public Window1() : base(WindowTheme.Green, true)
        {
            InitializeComponent();
        }

    }
}
```

For `WindowTheme.Red` theme:

```
namespace WindowsApplication1
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>

    public partial class Window1 : WindowBase
    {

        public Window1() : base(WindowTheme.Red, true)
        {
            InitializeComponent();
        }

    }
```

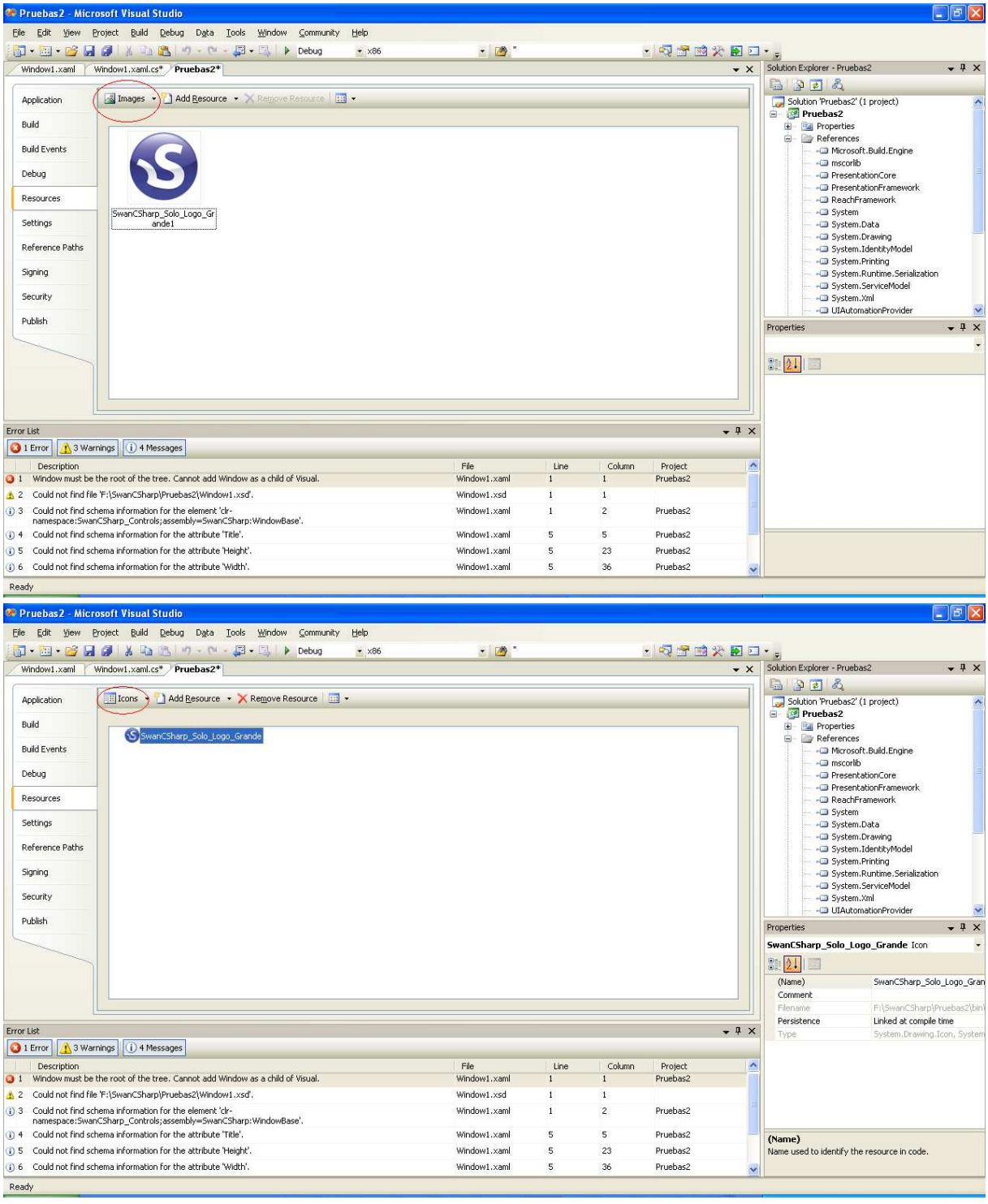
```
}
```

Important Note: In versions of Windows Presentation Foundation (WPF) included in the .NET Framework, up to date of publication of this document, **DO NOT ALLOW** WPF windows inherit because XAML files are not inheritable. The WindowBase inheritance is correct, but the form can NEVER be previewed in the IDE, and the objects on it will have to be placed by the methods described later in this document.

11.1.3 Images and icons in Windows "SwanCSharp"

Many methods in the namespace "SwanCSharp_Controls" allow use pictures and / or icons, and these files are recommended to be integrated into our development projects through the resource file by Visual Studio project. This will integrate with our compiled and can be easily called from the code to be passed as parameter.

In the picture below you can see two sample screens: The first one where images are integrated in the resource file, and the latter which integrates icons in the resource file.



11.1.4 Property “MainGrid”

In class "WindowBase" there is a property of type "Grid" call "MainGrid" that will be highly relevant in the development of a window "SwanCSharp": Through Property "MainGrid" we will be able to place all controls and objects to use in each form.

11.1.5 Initial properties of “SwanCSharp” window

There are a set of initial properties that can be set in a window "SwanCSharp", these initial properties determine the graphical display of the window.

11.1.5.1 Window title

The title of the window will be set by the method SetWindowTitle of WindowBase class and within the constructor of the window XAML created (in our example Window1.xaml.cs). As the first parameter is passed the text to be displayed as the title of the window; in the second parameter reports the desired colour for the text, a feature that can not be applied on a standard WinForm. An example may be:

```
public Window1() : base(false)
{
     InitializeComponent();
     SetWindowTitle("Test of SwanCSharp", Brushes.White);
}
```

With the line "SetWindowTitle" included under "InitializeComponent" the window to be displayed when running the solution is as follows:

11.1.5.2 Window icon

The icon of the window will be set by the method SetIconWindowTitle WindowBase class in the constructor of the window XAML created (in our example Window1.xaml.cs). There are two overloads for this method, the first parameter is passed as an object "Bitmap", and the second overload is passed as a parameter an object "Icon". In this way we can establish as an icon of the window one standard bitmap image or an ICO file (unlike the base form in .Net that only supports ICO files). The icon will always be displayed in a size of 32x32. An example may be:

```
public Window1() : base(false)
{
```

```
        InitializeComponent();
        SetWindowTitle("Test of SwanCSharp", Brushes.White);
        SetIconWindowTitle(Properties.Resources.SwanCSharp_Solo_Logo_Grande);
}
```

You can check our sample image has been included in the project's resource file, as explained in "Images and Icons in windows SwanCSharp" included earlier in this document.

With the line "SetIconWindowTitle" included under "InitializeComponent" the window to be displayed when running the solution is as follows:

11.1.5.3 Control-Box buttons

The WindowBase's control buttons (close, maximize, minimize, and notify) are not displayed if not executed expressly methods, and this allows us complete freedom to define the buttons, unlike standard developments in .Net that only allow to disable all buttons at once (what prevents us from setting an icon in the title bar), or independently off maximize and minimize icons unable to hide the close button. With WindowBase class we will be able to deactivate the three separate buttons without losing other resources, using the methods 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();
}
```

In the above example you enable the window to show the three buttons, but if we remove any of the methods, the window is displayed without the corresponding button can hide all three simultaneously without losing the ability to include the icon in the toolbar title. The buttons are displayed from right to left in the order in which they execute their methods, and

this allows us to alter the natural order of those buttons. The window to be displayed when running the solution is as follows:

The button "Notify" simply shows a button that runs the "Hide()" command, and is very useful, for example, for applications such as "Tray application".

11.1.5.4 "Moveable" window

One feature missing since Visual Basic 6.0 is the property "Moveable" of windows could block drag the window around the screen of Windows, being fixed in the start position. In either .Net version don't exists property "Moveable" in the forms (to block drag a window must develop custom classes), but in our "WindowBase" class there is a method called IsMoveable that enables / disables drag the window.

```
public Window1() : base(false)
{
	InitializeComponent();

	IsMoveable(true);
}
```

In the above example the window can be dragged and change position into Windows screen.

```
public Window1() : base(false)
{
	InitializeComponent();

	IsMoveable(false);
}
```

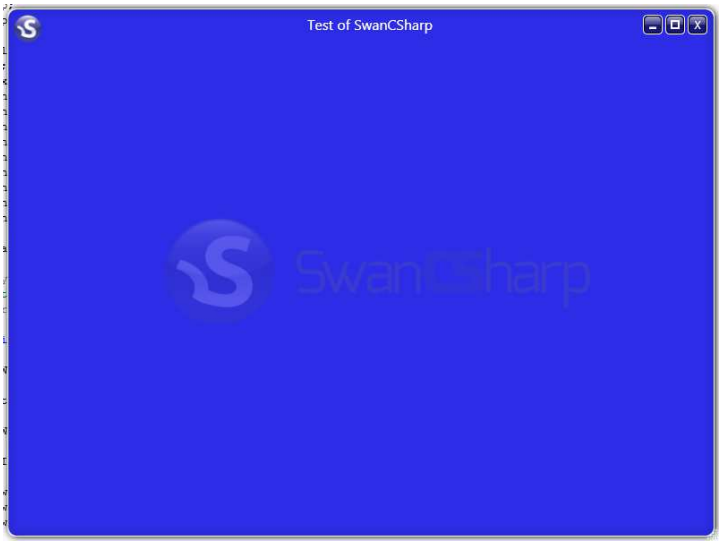
In the above example the window cannot be dragged and change position into Windows screen.

11.1.5.5 Watermark

In the forms "SwanCSharp" we can include a watermark in the background form with a particular image. In class "WindowBase" there is a method called "ShowWaterMark" that allows us to include the watermark; needs eight parameters which are: Bitmap image, the image width, height, the initial position (using the enumerator WaterMarkStartPosition existing in WindowBase class), and the four values to set a margin (left, top, right, bottom) that allow us to move freely the watermark. An example may be:

```
public Window1() : base(false)
{
     InitializeComponent();

     SetIconWindowTitle(Properties.Resources.SwanCSharp_Solo_Logo_Grande1);

     SetWindowTitle("Test of SwanCSharp", Brushes.White);

     ShowCloseButton();
     ShowMaximizeButton();
     ShowMinimizeButton();

     ShowWaterMark(Properties.Resources.SwanCSharp_Nuevo_Grande, 512, 190,
WaterMarkStartPosition.Center, 0, 0, 0, 0);
}
```

In the above example we use an image loaded as a resource in the project C#, and it provides that the starting position is the center of the form, we send the four margin values with "zero" because it does not want to move that watermark initial center position. The resulting form is:



11.1.6 Controls and objects for windows “SwanCSharp”

In this section we will describe a set of controls and objects created exclusively for windows "SwanCSharp", we can use freely in our developments.

Needless to say that all the functions defined in this section return a control type object or a set of controls (StackPanel). If we want to create these controls in our "SwanCSharp"

windows we create the objects as module level variable so that we can at any time access its properties and methods. Eg:

```
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")
        {
        }
   }
}
```

In the previous example we declare control "mtxtTextBox" as module level variable, so we can access its methods and properties from any window procedure "Window1".

However, if the object returned by the custom control is not a specific control but a collection of controls (StackPanel), we create the control group at the module level, and we can access its child elements to reach their methods and properties . Eg:

```
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();
    }
}
```

In the previous example declare StackPanel control "mobjUser" as module level variable, so we can access its methods and properties from any procedure of "Window1". Later "SwancSharp" created in the StackPanel a textbox and a label, therefore the "child" elements of "mobjUser" are: "0" (TextBlock) and "1" (TextBox). We want to access the method "Focus ()" of the TextBox (second element of StackPanel), and for this example we use the method "Children" of main object "StackPanel".

When we use StackPanel objects may not have access to all methods of each “child” object, for example cannot directly access the "Text" property of an object "TextBox" using the "children" of "StackPanel". For example if we have a StackPanel "stkUser" whose first element "child" is a "TextBlock" and second element is a "TextBox", if we clean the box can do:

```
TextBox lobjUser = (TextBox)stkUser.Children[1];
lobjUser.Text = "";
```

Or we can use "SetValue":

```
stkUser.Children[1].SetValue(TextBox.TextProperty, "");
```

Sometimes we can find any particular case in which SetValue unusable. For example if we have a StackPanel "stkPassword" whose first element "child" is a "TextBlock" and second element is a "PasswordBox" if we clear the text box can not access the property "Password" from "SetValue" by therefore we can only clean by:

```
PasswordBox lobjPassword = (PasswordBox)stkPassword.Children[1];
lobjPassword.Password = "";
```

11.1.6.1 Command button

In class "WindowBase" we have a control "Button" with personalized style according to the window "SwanCSharp". There are two functions that return a custom "Button" object: "CreateWindowButton" and "CreateWindowButtonWithImage". The first function create buttons that only show text, and need nine parameters: the text to display in the button (content), the name of the control, the width of the button, the height of button, and the following 4 parameters are applied margins (left, top, right, bottom) to fit the button at the desired location within the form; in the last parameter is passed the object "RoutedEventHandler" that contains the procedure where there will be the process when clicking on the button. In the second function (CreateWindowButtonWithImage) shows text and an image on the button, and have the same parameters as the first, but adding other three parameters to define the image to be inserted and its width and height. Within this there are two overloads for second function, in the first we pass a "Bitmap", and the second we can pass an "Icon" object.

Below we show an example using the "CreateWindowButton", excluding an image on the display of the button.

```
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
    */
}
```

In the code we can see that we call the function "CreateWindowButton" which returns a "Button" control. Previously we created the "cmdExecute_Click" with the code to execute when the button is pressed. The last line add the "Button" object on property "MainGrid" of our form. The result is as follows:

Below we show an example using CreateWindowButtonWithImage function, including icon (ICO) in the display of the button (the ICO file is included in the resources of the sample C# project).

```
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
    */
}
```

In the code we can see that we call the function "CreateWindowButtonWithImage" which returns control "Button". Previously we created the "cmdExecute_Click" with the code to execute when the button is pressed. The last line gives the object "Button" on property "MainGrid" of our form. The result is as follows:

11.1.6.2 Check Box

In "WindowBase" class we have a “CheckBox” custom control style according to the "SwanCSharp" window. There is a function that returns a custom "CheckBox": "CreateWindowCheckBox" that includes a total of 9 parameters. The text to display in the control (content), the name of the control, width, height, and the 4 following parameters are applied margins (left, top, right, bottom) to position the control in the desired location within the form, in the last parameter indicates whether you want a "shadow" effect for the form.

Below we show an example using the "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);
}
```

The result is as follows:

11.1.6.3 Main menu

In class "WindowBase" have a custom control that allows us to include an options menu (Pop-Up) with all the necessary features, allowing also include some special features, such as an icon for the horizontal main menu, and to allow including graphics icons in the vertical "Pop-Up" menu.
To create a menu in a "SwanCSharp" window let's use the "CreateWindowMenus" which will be added to the object MainGrid.
The “SwanCSharp” menu allows us to create a first row of horizontal menu options that open windows "Popup" with vertical menu options, allowing recursively add all desired options and submenus.
It also allows the inclusion of graphic icons to the left of each menu option, allowing those icons also in the main menu options horizontally. The function allows you to enable / disable icons display in vertical and horizontal options separately.

The first step in creating a menu is to create an object of type "MenuOptions" that exists in the namespace "SwanCSharp_Controls", the object "MenuOptions" is a structure of five variables: The first variable "NameMainMenuControl" which is the name defined internal control, the second variable is "NameToDisplayMainMenu" which is the name of the menu

item to be displayed on screen, the third variable is "IconToDisplay" which load a graphic icon with an “icon” object, the fourth variable is "SubmenuOptions" which is a new object "MenuOptions" we can create to include in the menu we are creating a second menu that depend on it, by this fourth variable that we want to create submenus recursively, the fifth variable is "MenuClickEvent "which is a variable that inform an object "RoutedEventHandler", which is the procedure that will run when you press on the menu (click event).

We will display a menu of example, and for this we will imagine that we want an option menu with a horizontal option called "File" that clicking on it will show us a window "Popup" with a submenu option called "New".
The first step is to identify how many options we will have to manage a procedure to be run when clicked. In our example in the "File" is not going to handle a click, as it shows the submenu "New", therefore we do not need an event "RoutedEventHandler". In contrast to the "New" if we need a procedure to include the code to execute when clicked. The first step is to create the object "RouterEventHandler" to the "New":

```
private void mnuNew_Click(object sender, RoutedEventArgs e)
{

}
```

Within the constructor of the window "Window1" let's create object "MenuOptions" that will define the menu. First create the vertical main menu "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;
}
```

In the source code above you can see that we create an array of "MenuOption" with a single element, as in the horizontal menu only have the "File". To use the graphic icon Floppy_Drive we previously loaded as a project resource type "Icon". The next step is to create a second object "MenuOption" different to the submenu "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;
```

In the previous source code looks like interact the procedure "mnuNew_Click" with the click event of this menu item (New). Now we want the "New" is a submenu "Popup" to open when you click on "File". So the object "lobjSubmenuFileOption" becomes an object of "lobjMenuOptions".

```
lobjMenuOptions[0].SubMenuOptions = lobjSubMenuFileOption;
```

We have already constructed the object "MenuOptions" to create the menu, the next step is to call the method that will build the needed object and then add it to the "MainGrid" our "SwanCSharp" window:

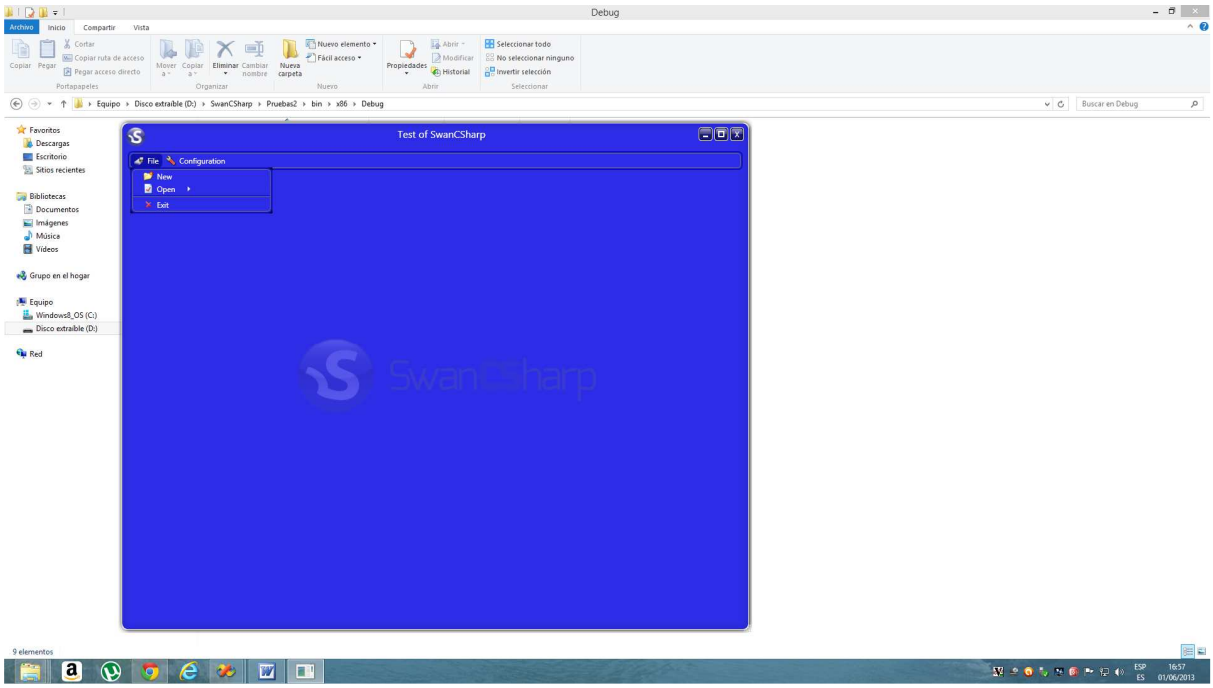
```
Menu lobjMenu = CreateWindowMenus(lobjMenuOptions, true, true);
MainGrid.Children.Add(lobjMenu);
```

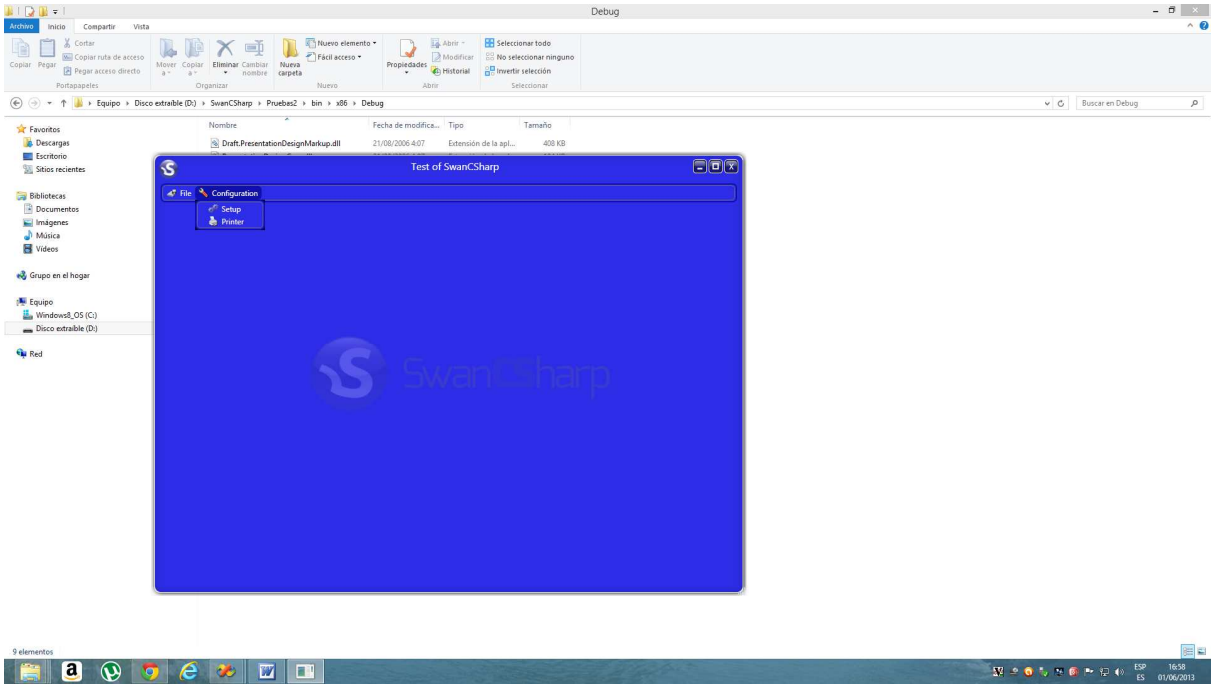
In the first parameter of "CreateWindowMenus" pass the object "MenuOption" created, in the second parameter will indicate whether we wish to show the icons in the main horizontal menu, and the third parameter passed if we want to show icons on all vertical options menu. The complete code for the window looks like this:

```
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.
             */
        }
    }
}
```

Create a "SwanCSharp" menu is based on creating "MenuOptions" objects that are added to the main structure recursively to finish by calling the function "CreateWindowMenus" that will return the "Menu" object built and added to the "MainGrid" of our window. Below we show an example of more complete menu:






```
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

In class "WindowBase" we have a control "ListBox" style personalized according to the window "SwanCSharp". There are two functions that return "ListBox" and "StackPanel" custom objects to us (as chosen function): "CreateWindowListBox" and "CreateWindowListBoxWithLabel". Both functions allow us to select if you want a shadow effect to control the size, etc.

The first function (CreateWindowLabelBox) only creates the text box, and receive a total of 8 parameters: The control name, the width of the control, the height of the control, and the following 4 parameters are the margins apply (left, top, right, bottom) to position the control at the desired location within the form, in the last parameter indicates whether to show an effect shade.

In the second function (CreateWindowListBoxWithLabel) create a StackPanel that integrates text label to display and ListBox, all in a single object. The function consists of thirteen parameters: the first eight parameters are to define the "ListBox" and the last five parameters are to define the "Labelbox". The first eight parameters are exactly the same as in the "CreateWindowListBox", the last five parameters are: The name that will be in control, the text to display on the label, the height of the control, the text colour of the label and alignment of the text within the label.

Is included below the source code for an example of on list box:

```
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 Text box

In class "WindowBase" we have a control "TextBox" style personalized according to the window "SwanCSharp". There are two functions that return "TextBox" and "StackPanel" custom objects to us (as chosen function): "CreateWindowTextBox" and "CreateWindowTextBoxWithLabel". Both functions allow us to select if you want a shadow effect to control the size, text alignment, etc.

The first function (CreateWindowTextBox) only creates the text box, and receive a total of 9 parameters: The control name, the width of the control, the height of the control, and the following 4 parameters are the margins apply (left, top, right, bottom) to position the control at the desired location within the form, in the eighth parameter is passed chosen text alignment, in the last parameter indicates whether to show an effect shade.

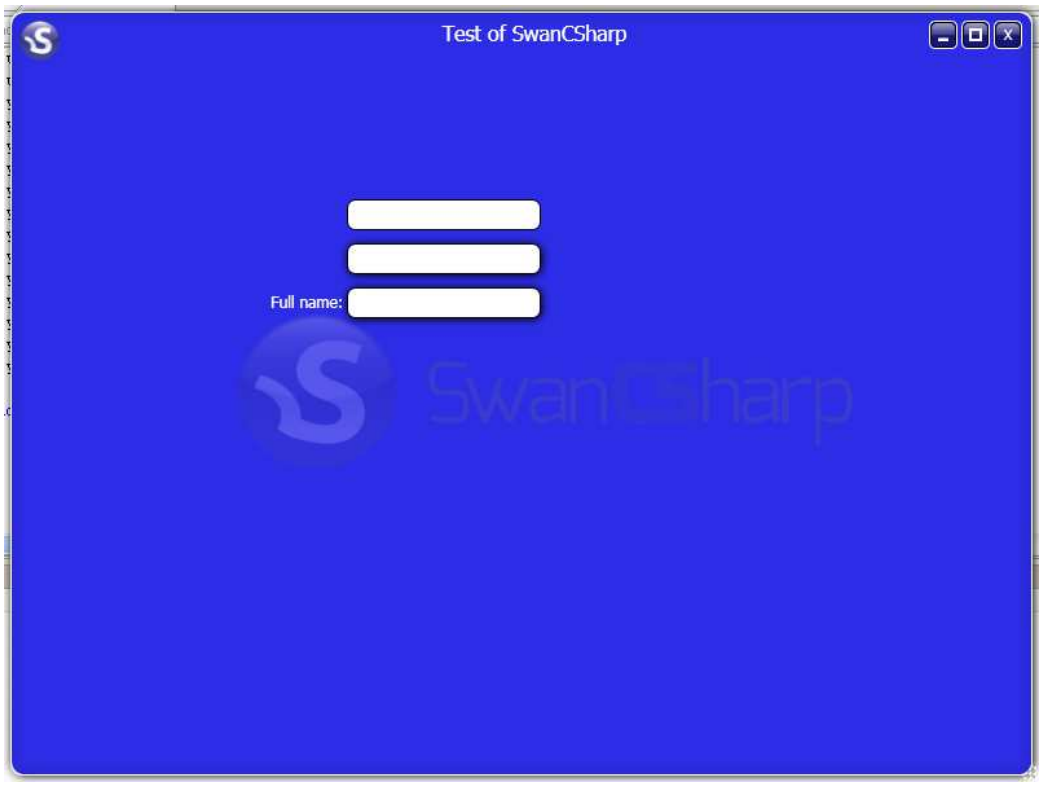
In the second function (CreateWindowTextBoxWithLabel) create a StackPanel that integrates text label to display and text box, all in a single object. The function consists of fourteen parameters: the first nine parameters are to define the "TextBox" and the last five parameters are to define the "Labelbox". The first nine parameters are exactly the same as in the "CreateWindowTextBox", the last five parameters are: The name that will be in control, the text to display on the label, the width of the control, the text colour of the label and alignment of the text within the label.

Is included below the source code for an example of three text boxes in various combinations (unlabeled and shade, shady unlabeled and labelled shaded):

```
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);
}
```

According to the above source code, to run the application, it would show on the screen the following "SwanCSharp" form:



11.1.6.3 PasswordBox

In "WindowBase" class we have a "PasswordBox" control custom style according to the window "SwanCSharp". There are two functions that return objects we "PasswordBox" and "StackPanel": "CreateWindowPasswordBox" and "CreateWindowPasswordBoxWithLabel". Both functions perform exactly the same of the functions “CreateWindowTextBox” and “CreateWindowTextBoxWithLabel" with the difference that instead of generating a standard “TextBox", those functions generate a special box for entering passwords.

11.1.6.1 LabelData

In "WindowBase" class we have a "LabelData" control customized style according to the window "SwanCSharp". The control creation function returns a "StackPanel".

The "CreateWindowLabelData" function is designed to create a control that displays text divided into two parts: Label and data. The idea is to display data in different color for label and data. Are passed a total of fifteen parameters: The name that will be in control, the height of the control, the size of the font, the following 4 parameters are applied margins (left, top, right, bottom) to position control in the desired location within the form, the text to display on the label, the label width, text alignment on the label, the color of the label text, the width of the data, the text alignment of the data, and the text color of the data.

Below we show the source code for an example of two controls "LabelData" on a form "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);}
```

According to the above source code, to run the application, it would show on the screen the following "SwanCSharp" form:

If you want to change any data during the execution of our development, we can do it this way:

```
TextBlock lobjText = (TextBlock) lobjStack.Children[1];
lobjText.Text = "111222";
```

11.1.6.2 GroupBox

In class "WindowBase" we have a "GroupBox" control with personalized style according to the window "SwanCSharp". There is a function that returns a custom "GroupBox": "CreateWindowGroupBox". The function has two overloads.

The first overload has three parameters: The name of the control, the caption text control, and an array of object "UIElement". This overload will create a “GroupBox” that covers the entire window, avoiding the title bar and the area reserved for the main menu.

The second overload has nine parameters: The name of the control, the caption text of the control, the control width, the control height, the following four parameters correspond to the "Margin", and an array of object "UIElement". This second overload allows us to fully define the "GroupBox" the size and location within the form.

The “GroupBox” control is used to contain other child controls, and to define the child controls contained in the "GroupBox" is used "UIElement" array. For example, if we place three text boxes inside the GroupBox, before array creating, we create a three-object "UIElement". Eg:

```
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;
```

Once you create the "UIElement" array with all the controls that will have the "GroupBox" inside, we create the “GroupBox” control and add it to the property "MainGrid" window:

```
GroupBox lobjGroupBox = new GroupBox();
lobjGroupBox = CreateWindowGroupBox("grpGroup", " Management ", 300, 200,
16, 85, 0, 0, lobjUIElements);

MainGrid.Children.Add(lobjGroupBox);
```

Following is the complete source code of the previous example that shows a "GroupBox" with three boxes of text at a specific form:

```
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);
}
```

According to the above source code, at run the application, it would show on the screen the following "SwanCSharp" form:

11.1.6.1 GroupLabelData

In "WindowBase" class have control "GroupLabelData" customized style according to the "SwanCSharp" window. There is a function that returns a "GroupBox" custom: "CreateWindowGroupLabelData".

The "CreateWindowGroupLabelData" function has thirteen parameters: The name of the control, the title of the control, the control width, the control height, the size of the font, the following four parameters correspond the "Margin", the height of the whole "LabelData", the width of the "label", the width of the "Data", and the object array "LabelData" that contains the information of each data tag line.

The first step is to create a "LabelData" array. In our case we will create three lines with corresponding data labels each label. E.g.:

```
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;
```

Once you create the "LabelData" array object with the information of each line, we create "GroupLabelData" control (which will be a GroupBox object) and add it to the "MainGrid" window property:

```
GroupBox lobjGroup = CreateWindowGroupLabelData("lobjGroup", "Label and
Data", 175, 140, 14, 100, 80, 0, 0, 24, 120, 100, lobjLabelData);

MainGrid.Children.Add(lobjGroup);
```

Below we show the complete source code of the previous example that shows a "GroupLabelData" text lines:

```
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);
}
```

According to the above source code, to run the application, it would show on the screen the following "SwanCSharp" form:

11.1.6.2 ComboBox

In class "WindowBase" we have a "ComboBox" control style personalized according to the window "SwanCSharp". There are two functions that return custom "ComboBox" and "StackPanel" objects (as chosen function): "CreateWindowComboBox" and "CreateWindowComboBoxWithLabel". Both functions allow you to select whether you want a shadow effect to control the size, text alignment, etc.

The first function (CreateWindowComboBox) only creates the control, and pass a total of eight parameters: The name that will be in control, control the width, the height of the control, and the next four parameters are the margins to apply (left, top, right, bottom) to position the control in the desired location within the form, the last parameter indicates whether to show an effect shade.

In the second function (CreateWindowComboBoxWithLabel) creates a StackPanel that integrates text label to display and control, all in a single object. The function consists of

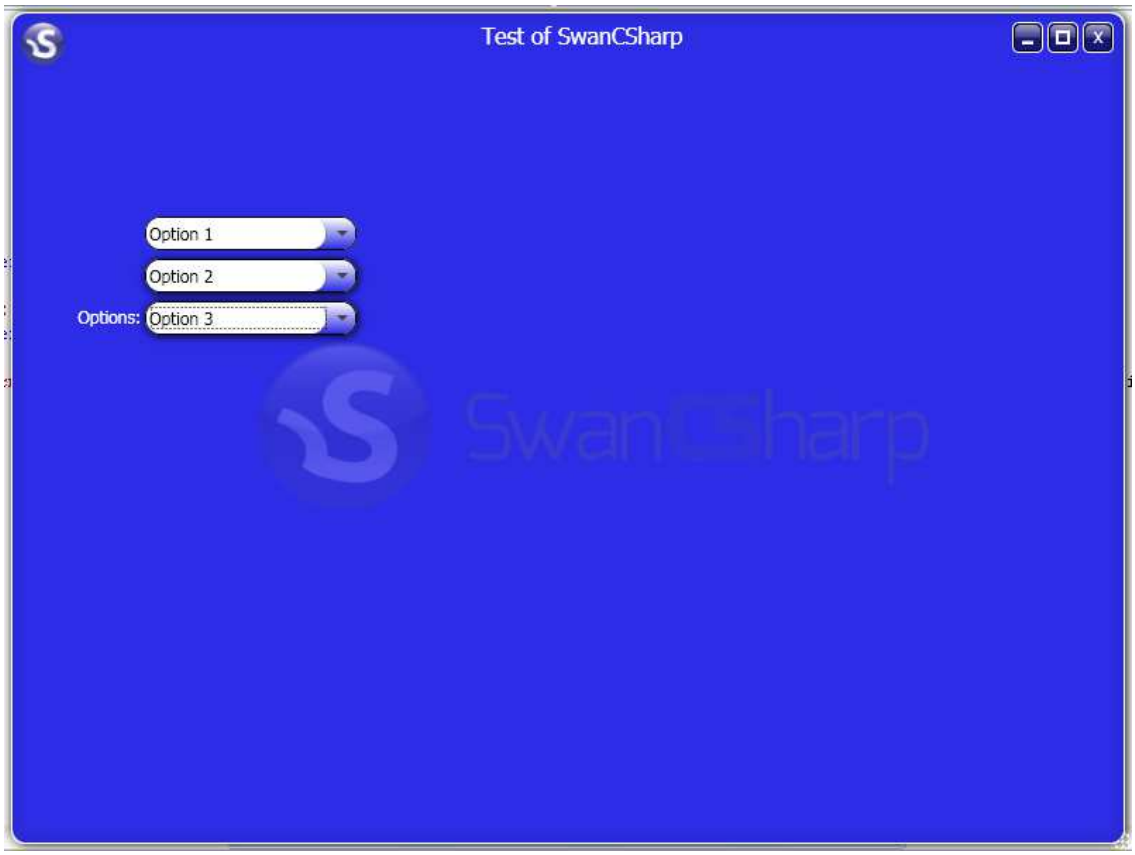
fifteen parameters: the first eight parameters to define the "ComboBox", the next two to load the data in the control, and the last five parameters to define the "Labelbox". The first eight parameters are exactly the same as in the "CreateWindowComboBox", the following two parameters are of type "string []" to pass the data and the index of each row to load (if you do not want to load data, passed both parameters as "null"), the last five parameters are: the name that will be in control, the text to display on the label, the width of the control, the text colour of the label and the text alignment within the label.

The following is the source code for an example of three ComboBox in various combinations (unlabeled and shade, shady unlabeled and labelled shaded):

```
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);
}
```

According to the above source code, at execute the application, it would show on the screen the following form "SwanCSharp":



11.1.6.3 DataGrid

In "WindowBase" class we have a "DataGrid" control custom style according to the window "SwanCSharp". There is a function that returns a "ListView" custom: "CreateWindowDataGrid". The function has nine parameters: The name of the control, the object "DataTable" that contain the columns to display and the data, the width of the control, the height of the control, the following four parameters correspond to "Margin ", and the last parameter indicates whether you want a shadow effect to control. An example of call control "DataGrid" can be:

```
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);
```

Is shown below the complete source code of the previous example that shows a "DataGrid" with existing data in a database "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);

}
```

According to the above source code, to run the application, it would show on the screen the following "SwanCSharp" window:

11.1.6.4 TabControl

In "WindowBase" class we have a "TabControl" control custom style according to the window "SwanCSharp". There is a function that returns a "TabControl" custom: "CreateWindowTabControl". The function has nine parameters: The name of the control, an array string object with the items, the width of the control, the height of the control, the following four parameters correspond to "Margin ", and the last parameter indicates whether you want a shadow effect to control. An example of call control "TabControl" can be:

```
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);
```

Is shown below the complete source code of the previous example that shows a "DataGrid" with existing data in a database "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);

}
```

According to the above source code, to run the application, it would show on the screen the following "SwanCSharp" window:

11.2 SwanCSharp_Controls.WindowError

The "WindowError" class allows us to display an error window with the format and style of the "WindowBase" windows of this namespace "SwanCSharp". A source code example that shows a "WindowError" window is as follows:

```
WindowError lobjError = new WindowError("Error", "Test of error message.");
lobjError.ShowDialog();
lobjError.Close();
```

When you run the above example source code are displayed the following window:

Optionally there exists a third parameter that allows us to choose the main colour of the window, chosen from the six existing themes.

11.3 SwanCSharp_Controls.WindowInformation

The "WindowInformation" class allows us to display an information window with the format and style of the "WindowBase" windows of this namespace "SwanCSharp". A source code example that shows a "WindowInformation" window is as follows:

```
WindowInformation lobjInformation = new WindowInformation("Information",
"Test of information message.");
lobjInformation.ShowDialog();
lobjInformation.Close();
```

When you run the above example source code are displayed the following window:

Optionally there exists a third parameter that allows us to choose the main colour of the window, chosen from the six existing themes.

11.4 SwanCSharp_Controls.WindowOKCancelQuestion

The "WindowOKCancelQuestion" class allows us to display a question window with the format and style of the "WindowBase" windows of this namespace "SwanCSharp". A source code example that shows a "WindowOKCancelQuestion" window is as follows:

```
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();
```

When you run the above example source code are displayed the following window:

Optionally there exists a fourth parameter that allows us to choose the main colour of the window, chosen from the six existing themes.

11.5 SwanCSharp_Controls.WindowWarning

The "WindowWarning" class allows us to display a warning window with the format and style of the "WindowBase" windows of this namespace "SwanCSharp". A source code example that shows a "WindowWarning" window is as follows:

```
WindowWarning lobjWarning = new WindowWarning("Warning", "Test of warning
message.");
lobjWarning.ShowDialog();
lobjWarning.Close();
```

When you run the above example source code are displayed the following window:

Optionally there exists a third parameter that allows us to choose the main colour of the window, chosen from the six existing themes.

11.6 SwanCSharp_Controls.WindowYesNoQuestion

The "WindowYesNoQuestion" class allows us to display a question window with the format and style of the "WindowBase" windows of this namespace "SwanCSharp". A source code example that shows a "WindowYesNoQuestion" window is as follows:

```
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();
```

When you run the above example source code are displayed the following window:

Optionally there exists a fourth parameter that allows us to choose the main colour of the window, chosen from the six existing themes.

11.7 WindowFile

This window will show in a tree structure all logical drives in the system and their associated folders, and files, allowing you to select a specific file (open file dialog box), and returning the full path selected. To call the file selection window write the following code:

```
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();
```

The window can be displayed in Spanish and English. The first parameter of the constructor we pass the desired mask (if left blank, it is assumed *. *) and the window return three properties with all values of the selected file (full file name, file name without path, path without file name). An example screen is:

Optionally there exists a third parameter that allows us to choose the main colour of the window, chosen from the six existing themes.

11.8 WindowFolder

This window will show in a tree structure all logical drives in the system and their associated folders, allowing you to select a folder or a particular unit and returning the full path selected. To call the folder selection window write the following code:

```
WindowFolder lobjFolder = new WindowFolder(InterfaceLanguage.English);
lobjFolder.Owner = this;
lobjFolder.ShowDialog();
string lstrPath = lobjFolder.SelectedFolder;
lobjFolder.Close();
```

The window can be displayed in Spanish and English, and according to the sample code in the variable "lstrPath" receive the selected path in the window. An example screen is:

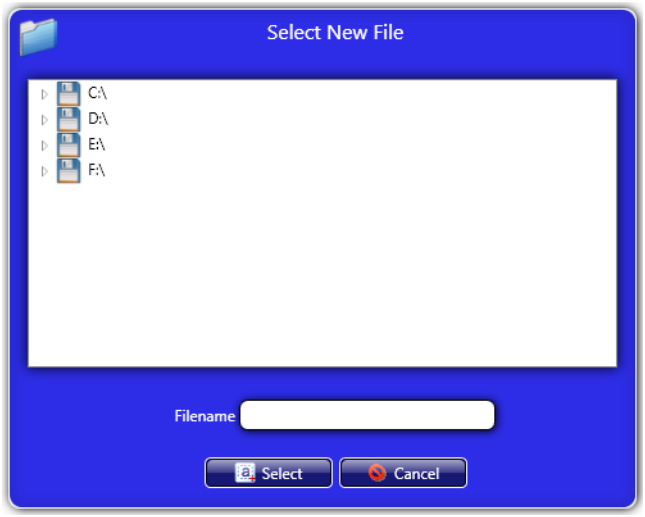
Optionally there exists a second parameter that allows us to choose the main colour of the window, chosen from the six existing themes.

11.9 WindowNewFile

This window will show in a tree structure all logical drives in the system and their associated folders, allowing build the "path" of a particular new file (dialog box to create files) and returning the full path. To call the file selection window write the following code:

```
WindowNewFile lobjFile = new WindowNewFile("swc",
InterfaceLanguage.English);
lobjFile.Owner = this;
lobjFile.ShowDialog();
string lstrNewFile = lobjFile.File;
lobjFile.Close();
```

The window can be displayed in Spanish and English. The first parameter of the constructor we pass the desired extension to the file name that the user enters on the screen (if you do not want any extension, indicate "String.Empty") we obtain a property (File) which returns the full path built using the selected path, the file name typed on the screen, and the selected extension to open the window. An example screen is:



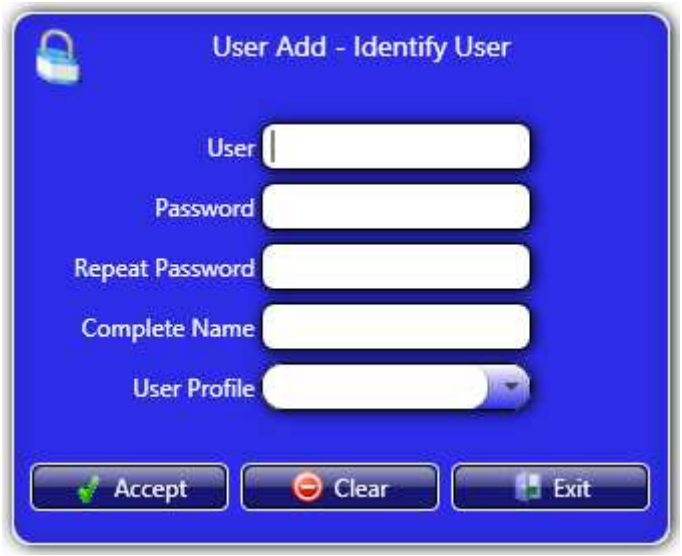
Optionally there exists a third parameter that allows us to choose the main colour of the window, chosen from the six existing themes.

11.10 WindowAddUser

This "Add User" window will allow us to add an option to register new users in “UserManagement” object. An example of the window called "Add User" is:

```
private void mnuAddUser_Click(object sender, EventArgs e)
{
      WindowAddUser lobjAddUser = new WindowAddUser(ref gobjUserManage,
SwanCSharp_Controls.InterfaceLanguage.English);
      lobjAddUser.ShowDialog();
      lobjAddUser.Close();
}
```

As the first parameter is passed the “UserManagementSQLServer” object (may also be Oracle, Firebird, MySQL, or Access), and the second parameter is passed the language of the user interface. Optionally there exists a third parameter that allows us to choose the main colour of the window, chosen from the six existing themes. An example screen is:



This screen is never going to allow a user profile "StandarUser" create another user. A profiled user "Administrator" only allows you to create another user "Administrator" or another user "StandarUser". For "SuperAdmin" user no restrictions whatsoever.

11.11 WindowLoginUser

The logical thing would when running the application will display a window "login" to verify a correct user entry. So before opening main window "Window1" we call the “Login” window. Then the source code would look like this for developments with the window "WindowBase" (is necessary to reference the SwanCSharp_Controls and SwanCSharp.Users namespaces):

```
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();
    }
}
```

The builder of this window has an overhead for each database managers supporting class "UserManagement" (SQL Server, Oracle, Firebird, MySQL, and Access). Windows forms of this class can be displayed in two languages through enumerated UserManagement.InterfaceLanguage, Spanish and English. Optionally there exists a third parameter that allows us to choose the main colour of the window, chosen from the six existing themes.

The above code will display a login window, if the user is not correct, the application will close. From the creation of the object "User Management" have a global variable "gobjUserManage" available throughout the application that has properties with the values of the user who accessed via login (UserName, UserPassword, UserCompleteName, Profile).

11.12 WindowPasswordUser

This window will allow us to change the password of the current user that is logged in at the time. This method opens a window where you will enter the current password, and will enter the new password. Pressing "OK" the password will be changed as long as the current password entered matches.

To call the password change window write the following code:

```
WindowPasswordUser lobjPassword = new WindowPasswordUser(ref
gobjUserManage, SwanCSharp_Controls.InterfaceLanguage.English);
lobjPassword.ShowDialog();
lobjPassword.Close();
```

As the first parameter is passed the “UserManagementSQLServer” object (may also be Oracle, Firebird, MySQL, or Access), and the second parameter is passed as the language

of the user interface. Optionally there exists a third parameter that allows us to choose the main colour of the window, chosen from the six existing themes. An example screen is:

11.13 WindowRemoveUser

This window will allow us to remove users from the database. To remove a user only need to enter his username. It is important to know that there are some rules to delete users: If the current user is "SuperUser" can delete any other user whatever their profile, if the current user's profile is "Administrator", can delete only "Standard User". The "Standard User" does not have permission to delete any other user whatever their profile.

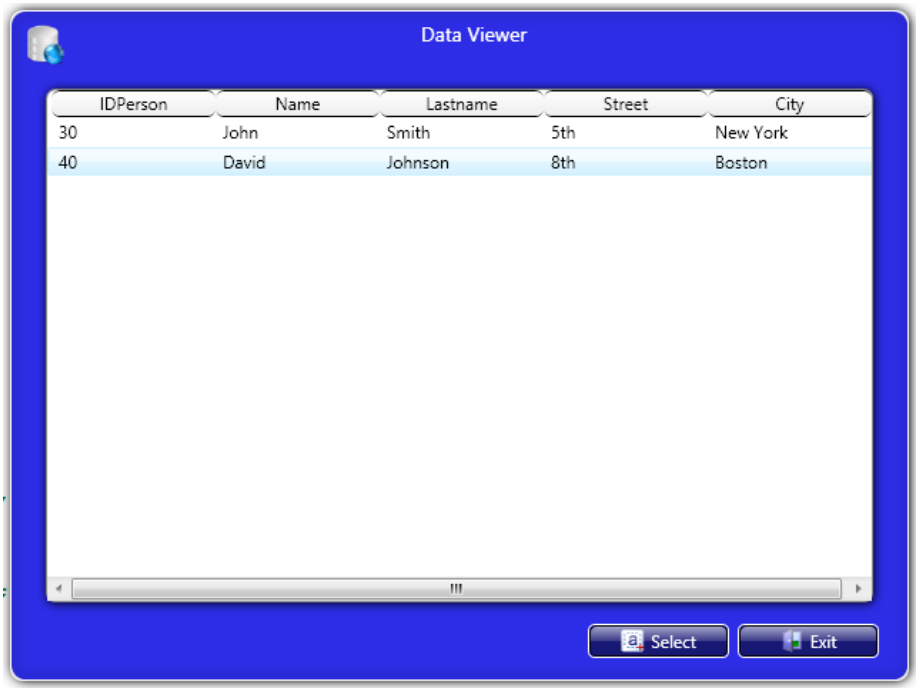
To invoke the user deleted window you write the following code:

```
WindowRemoveUser lobjRemove = new WindowRemoveUser(ref gobjUserManage,
SwanCSharp_Controls.InterfaceLanguage.English);
lobjRemove.ShowDialog();
lobjRemove.Close();
```

As the first parameter is passed the “UserManagementSQLServer” object (may also be Oracle, Firebird, MySQL, or Access) created for the application, and the second parameter is passed as the language of the user interface. Optionally there exists a third parameter that allows us to choose the main colour of the window, chosen from the six existing themes. An example screen is:

11.14 WindowDataQuery

The "WindowDataQuery" class allows us to display a window with a "grid" of data and returns a "DataRow" with the row that is selected in the window. A screen is as follows:



The "WindowDataQuery" class expects to receive four parameters which are: the DataTable with the data, the desired title for the form to be displayed, the desired title for the "frame" shown on the form, the size of the window (using the listed QueryWindowSize choosing from Small, Medium, High), and the language of the form (using the listed InterfaceLanguage, choosing between Spanish and English). Optionally there exists a fifth parameter that allows us to choose the main colour of the window, chosen from the six existing themes.

After running the function displays a Windows form showing the data on screen. The user can select a line by double-clicking on the row, or click on the row and clicking the "Select" button. The function returns a "DataRow" with the data of the selected row.

At the head of the reference procedure be added to the class:

```
using SwanCSharp_Controls;
```

A sample line of code can be:

```
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

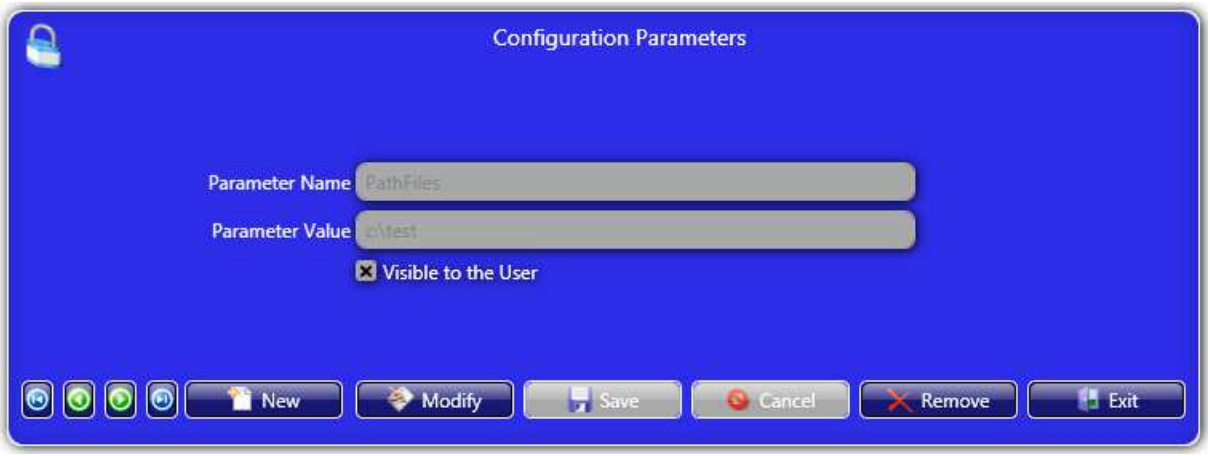
The "WindowParameters" will allow us to view and modify display all configuration parameters associated with our developments (existing five constructors, one for each data manager, SQL Server, Oracle, Firebird, MySQL, Access, and external file). The window can be displayed in two ways, one way for the administrator where you can add, modify, and delete all parameters (visible and invisible) which otherwise may query and modify only visible parameters, can not in any case create a new or delete a configuration parameter.

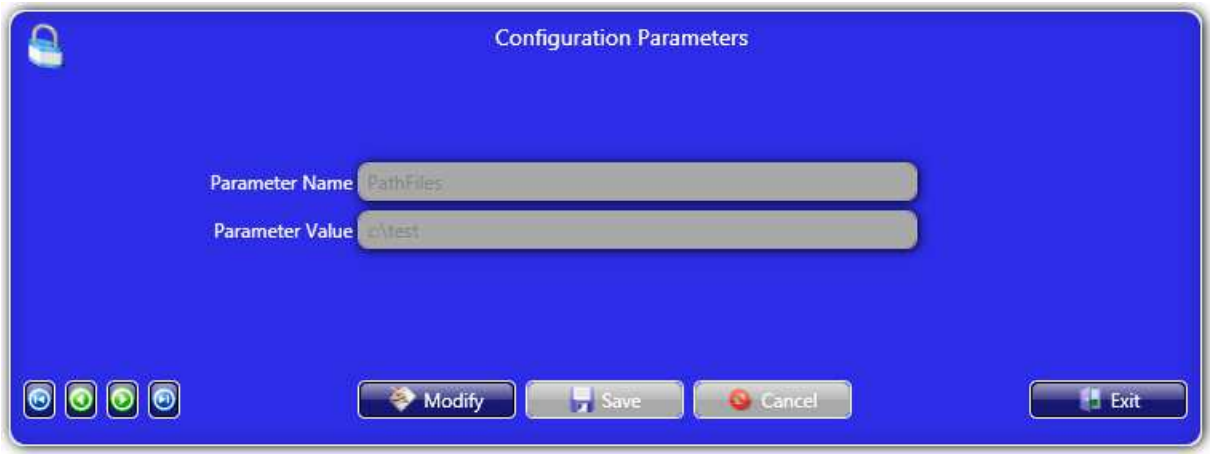
To call the parameter changes window with all the permissions we can write the following code:

```
gobjConfig = new ConfiguratorFile("config.cfg");
WindowParameters lobjParameters = new WindowParameters(gobjConfig, false,
true, InterfaceLanguage.English);
lobjParameters.ShowDialog();
lobjParameters.Close();
```

To call the parameter changes window to view and modify only the parameters "visible" we can write the following code:

```
gobjConfig = new ConfiguratorFile("config.cfg");
WindowParameters lobjParameters = new WindowParameters(gobjConfig, true,
false, InterfaceLanguage.English);
lobjParameters.ShowDialog();
lobjParameters.Close();
```





Optionally there exists a fifth parameter that allows us to choose the main colour of the window, chosen from the six existing themes.

11.16 WindowReportView

"WindowReportView" class allows us to show in a window any existing HTML report without rebuilding it. It is necessary to report the following parameters:

* Filename. - The name of the file you will when stored on disk.
* File path. - The disk path of the file.
* Title window. - The title to be displayed in the window.
* Size of the display window. - You can choose between Small, Medium, and High.
* Interface language screen. - We choose the language in which to display the report output.

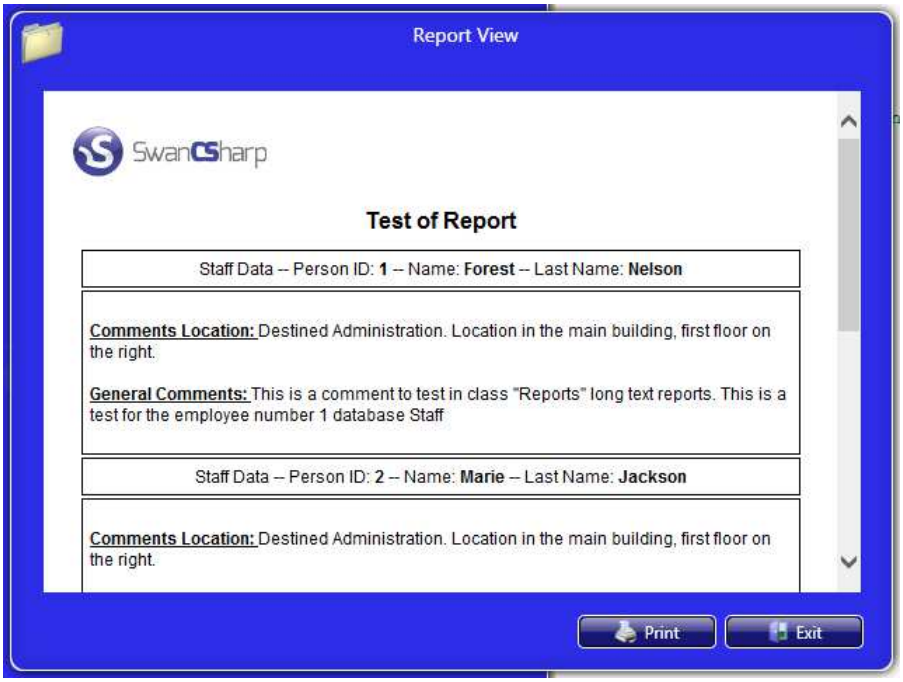
At the head of the reference procedure be added to the class:

```
using SwanCSharp_Controls;
```

A sample line of code can be:

```
WindowReportView lobjReport = new WindowReportView("report.html", "",
"Report View", SwanCSharp.Reporting.ReportViewWindowSize.Small,
InterfaceLanguage.English);
lobjReport.ShowDialog();
lobjReport.Close();
```

The previous source code show the following window:



Optionally there exists a sixth parameter that allows us to choose the main colour of the window, chosen from the six existing themes.

11.17 WindowSplash

The "WindowSplash" class allows us to display a welcome screen (splash) when run our application, or alternatively any window you want to display for a few seconds. The "WindowSplash" window displays a main logo in the center of the window, a secondary logo on any of the four corners, and horizontally centered text that can be adjusted vertically. The window can be displayed in any of the six existing color themes. The parameters of the window are:

* Text.- Text to display that will centered on the horizontal. If you do not want text we reported with "empty".
* Margin top.- The vertical distance which you want to position the text.
* Width.- The width of the “Splash” window.
* Height.- The height of the “Splash” window.
* Main image.- Image to be displayed in the center of the "Splash" window.
* Secondary Image.- Secondary image to be displayed in the center of the window "Splash". If you do not want, report with "null".
* Secondary image position.- Corner of the window "Splash" which will show the secondary image.
* Time to close.- Time (in seconds) that the window will remain open "Splash", until its closing.
* Transparency.- If desired or no transparency in the "Splash" window.
* Colour (Theme).- Main color of the "Splash" window.

At the head of the reference procedure be added to the class:

```
using SwanCSharp_Controls;
```

To create the "Splash" window, you use the commands:

```
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();
```

To execute the welcome screen just before the start of the application, it is recommended to place the source code in the constructor before the main window of our application. The previous source code would show the following window:

11.18 SwanCSharp_Controls.ClipboardAgent

The ClipboardAgent class allows us to capture all that is cut or copied to the Windows clipboard.

In the header of the referenced method be added the class:

```
using SwanCSharp_Controls;
```

To create the "ClipboardAgent" object does the following:

```
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);
```

Not only are we creating the object, but we are also creating an event for each type of data that the clipboard is able to capture, in total four events, so you have to create subroutines for each event:

```
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)
{

}
```

In the desired part of our source code we started the agent:

```
lobjClipAgent.StartAgent();
```

When you close your application, or on that part of our software it is considered appropriate, we should close the agent:

```
lobjClipAgent.CloseAgent();
```

11.19 SwanCSharp_Controls.GlobalHotKeys

The GlobalHotKeys class allows us to manage hotkeys globally (operating system). We can assign to the operating system a keystroke and then run a process each time it was pressed.

In the header of the referenced method be added the class:

```
using SwanCSharp_Controls;
```

To create the "GlobalHotKeys" object does the following:

```
GlobalHotKeys mobjHotKey = new GlobalHotKeys(this);

mobjHotKey.HotKeyIsPressed += new
GlobalHotKeys.HotKeyPressedEventHandler(mobjHotKey_HotKeyIsPressed);

mobjHotKey.StartHotKey(Key.LeftAlt, 0x42);
```

We're not just creating the object, but we are also creating an event will be fired when the keys are pressed. By "StartHotKey" method we started listening according the given parameters. The first parameter we determine if we press auxiliary key (can be `Key.LeftAlt`, `Key.LeftCtrl`, `Key.RightCtrl`, `Key.LeftShift`, `Key.RightShift`), and the second parameter the ASCII hexadecimal value of the desired key. In our example we passed "0x42" because the ASCII hexadecimal value 42 is the decimal value 66, which is the letter "B" corresponds. So in the above example we are checking the keystrokes the combination "Alt + B". If you do not wish to include an auxiliary key, we report the first parameter with the value "0".

The last step is to create the method that will be called by the event class:

```
private void mobjHotKey_HotKeyIsPressed(Key pobjAuxKey, int pintKey)
{
}
```

When you close your application, or part of our software it is considered appropriate, we must stop the HotKey object.

```
mobjHotKey.StopHotKey();
```

11.20 SwanCSharp_Controls.SW_Miscellaneous

The SW_Miscellaneous class cover those SwanCSharp.Miscellaneous generic functions that need customization to be executed in a window SwanCSharp.WindowBase.

11.20.1 LoadDataInComboBox

This procedure allows us to load in a ComboBox (generated from WindowBase class) all desired items, inserting Text and Value parameters desired (something not directly allow the ComboBox). You would create an array of string with the parameter "Text" for each item, and another array of "String" with the parameter "Value" of each item. The third parameter is passed by reference "ComboBox" we want to load.

In the header of the referenced method be added the class:

```
using SwanCSharp_Controls;
```

A sample line of code can be:

```
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);
}
```