Table of Contents

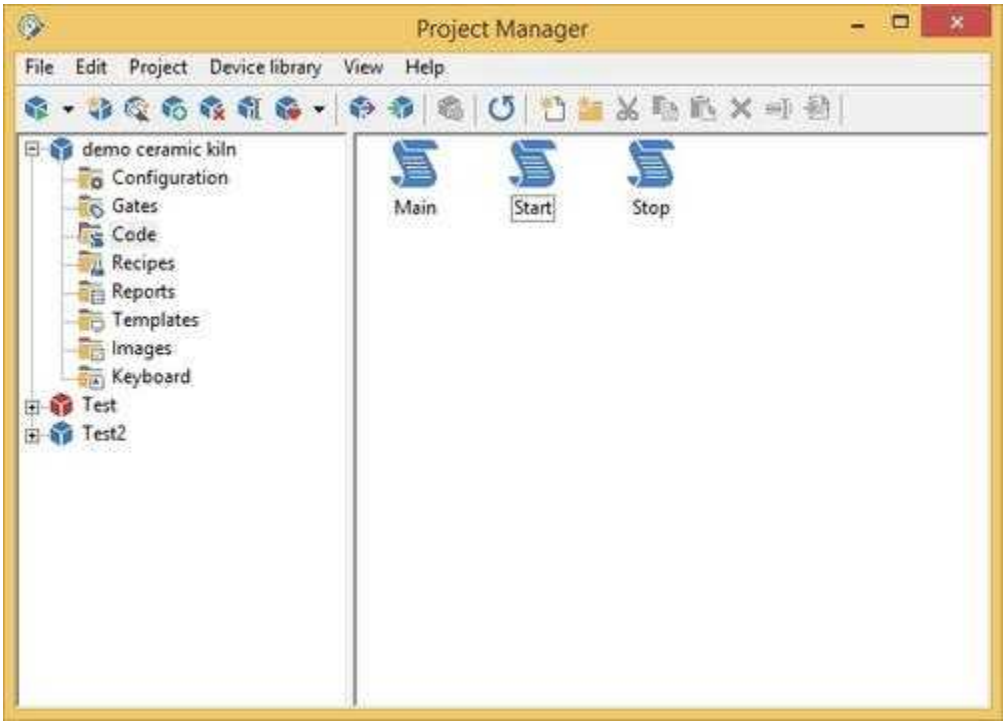
1 Introduction

Often, in a supervision project, comes the need to carry out special operations to the fulfillment of certain conditions or on operator request. Some examples include performing a calculation of a complex formula, send email or SMS, read or write text files, operate directly on gates value, manipulate certain objects of the synoptic displayed at that time, etc.
Code Builder is an editor designed to help you write these functions. It allows you to write pages of code and check its syntax; It offers a range of search tools and text manipulation typical of editors; furthermore, to simplify and reduce the presence of errors, the sequences of characters are highlighted with a different color in relation to their role in the syntax.

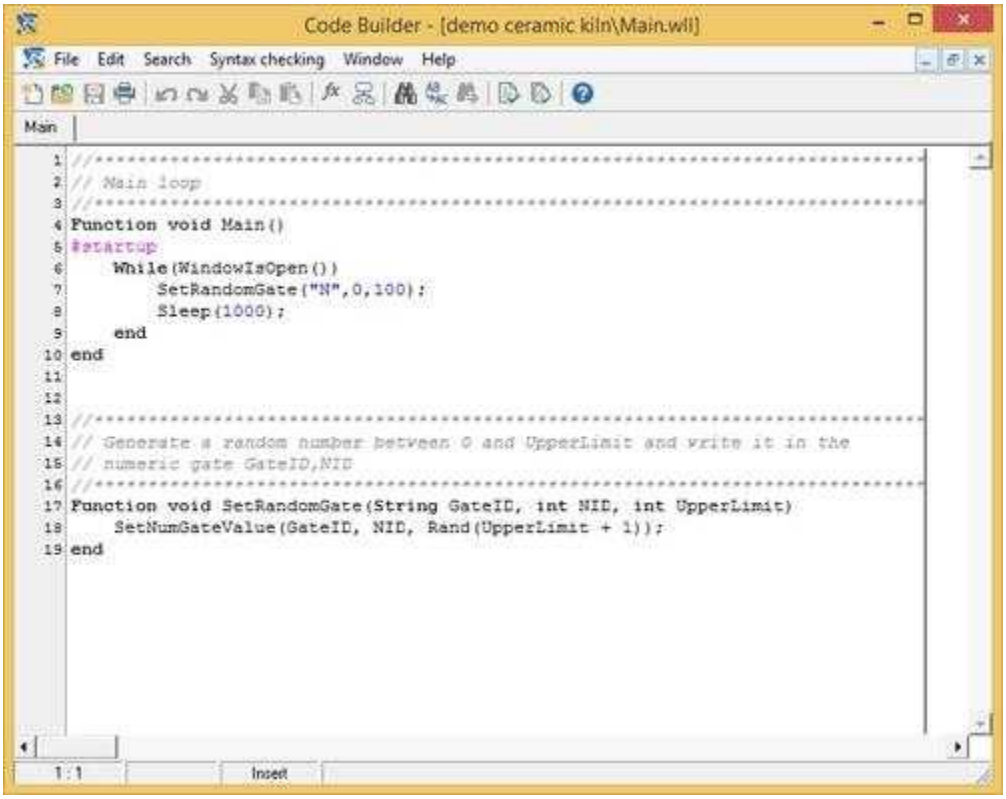
2 The first step

The creation of a new blank code file is performed in the *Project Manager*, by selecting the *Code* folder and then moving the mouse on the right window and pressing the right button: the *"New* | *File code"* menu entry is then displayed

Code Builder is started automatically by the *Project Manager* by double clicking on the icon to any page of language previously created.



Below you see Code Builder opened and in the workspace you can see a window titled Main.wll containing the file just uploaded .



The integrated environment consists of menus, speed buttons, a status bar for the information (at the

bottom) and an empty workspace.

The code windows on which to work will be shown in the workspace.

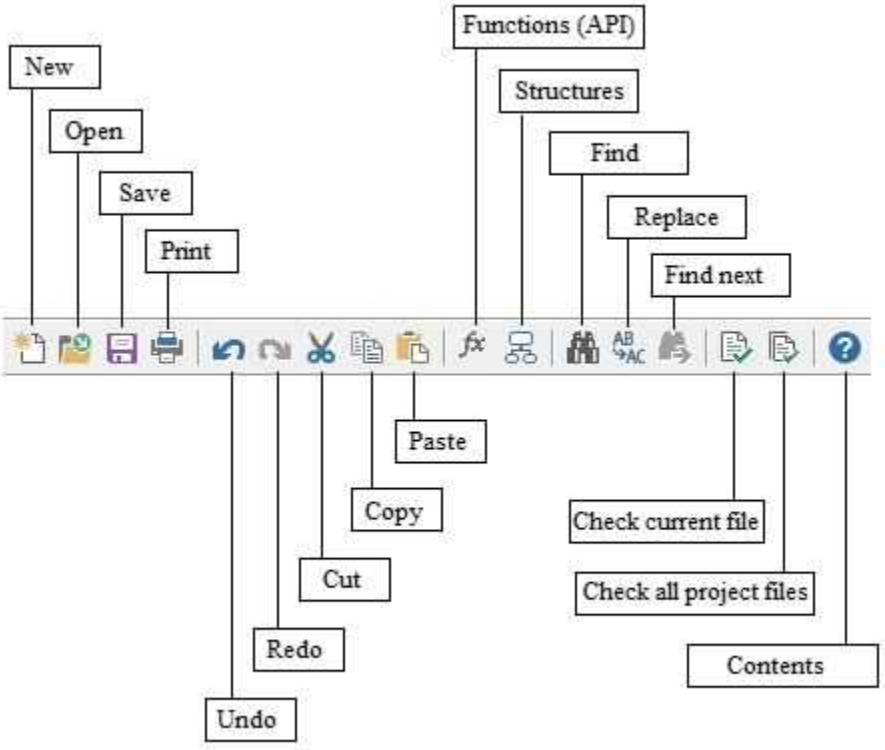
Note, in the new window, the different color of the example code. The text is colored according to the analysis made by the internal recognizer of *Code Builder* that categorizes the text elements. Of course you can change the attributes of every category at will, while the rules differentiating the classes of elements can not be changed and have their origin in the language syntax.

You may note, on the status bar, some elements that were not there before; they are the cursor position (row : column) in the first rectangle, and the word *insertion* in the third rectangle; this word indicates that every character typed with the keyboard will be inserted in the present position of the cursor. You can also decide to replace the characters instead of inserting them. Pressing the INS key you will note two changes: the cursor shape is wider and on the bar there is the word *overwriting*. This way the characters are no longer inserted in the text but they overwrite it; to go back to the insertion mode just press again INS.

3 Speedbar

The speedbar (figure below) has speed buttons for functions frequently called up from the menus.

By placing the mouse pointer on one of these buttons, a brief explanation of the function will be displayed.



4 Advanced use of the keyboard

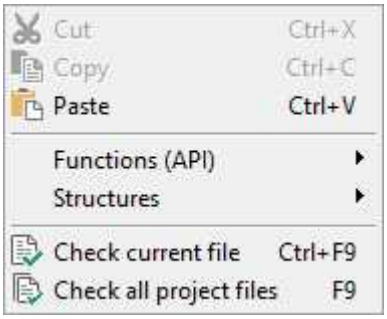
With the keyboard it is possible to access several functions provided by the menus; we have already seen that beside some menu items there is a combination of keys.

However, there are functions that can be accessed only through the keyboard: in the table there is a list of keys and their function.

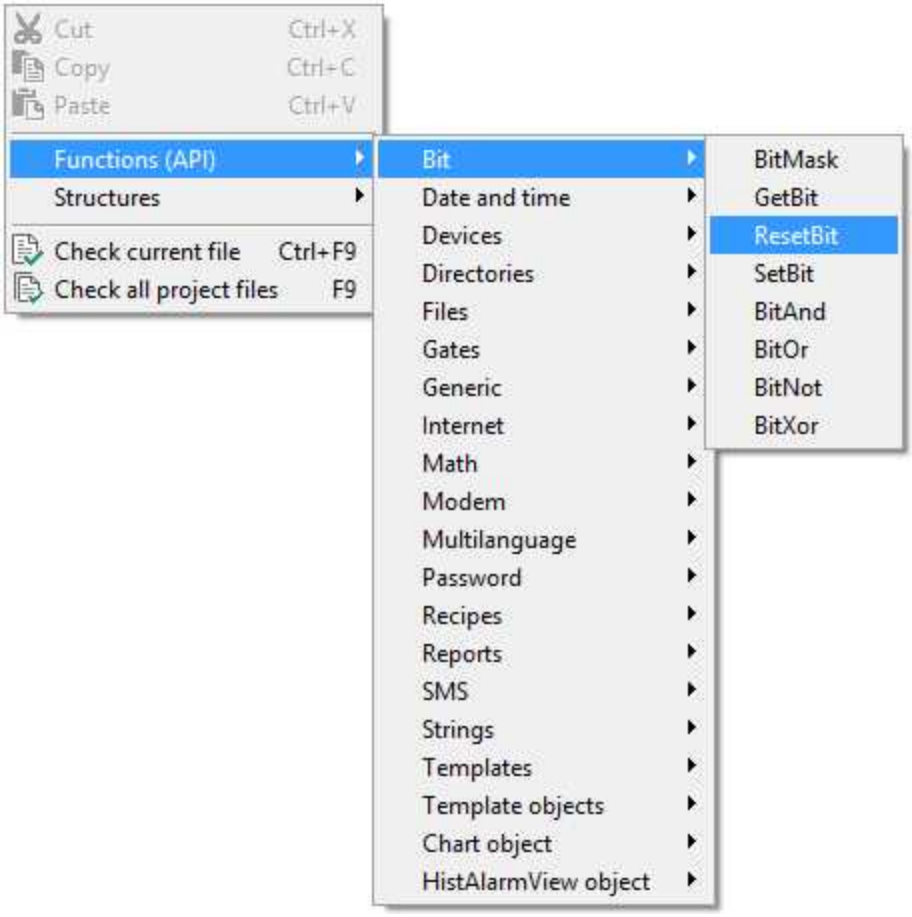
Keys	Function	Available with keystrokes	Available in the menu
CTRL + N	creates a new code window	✓	✓
CTRL + O	opens a *WinLog* code file	✓	✓
CTRL + S	saves the working file	✓	✓
CTRL + Z	undo	✓	✓
CTRL + C	copy to clipboard the selected text	✓	✓
CTRL + X	deletes the selected text	✓	✓
CTRL + V	pastes from the clipboard	✓	✓
CTRL + F	search in the text	✓	✓
CTRL + R	search and replaces the text	✓	✓
CTRL + F9	checks the syntax of the working code	✓	✓
F3	repeats the last search	✓	✓
F9	checks the syntax of the entire projects	✓	✓
F1	calls up the contextual help	✓	✓
INS	changes the insert state	✓	✗
CTRL+ Y	deletes the row at the cursor	✓	✗
ALT + N	brings up the next code window	✓	✗
ALT + P	brings up the previous code window	✓	✗
CTRL+ LEFT	move the cursor to the next left word	✓	✗
CTRL +RIGHT	move the cursor to the next right word	✓	✗
CTRL + HOME	move the cursor to the first line	✓	✗
CTRL + END	move the cursor to the last line	✓	✗

5 The use of the mouse

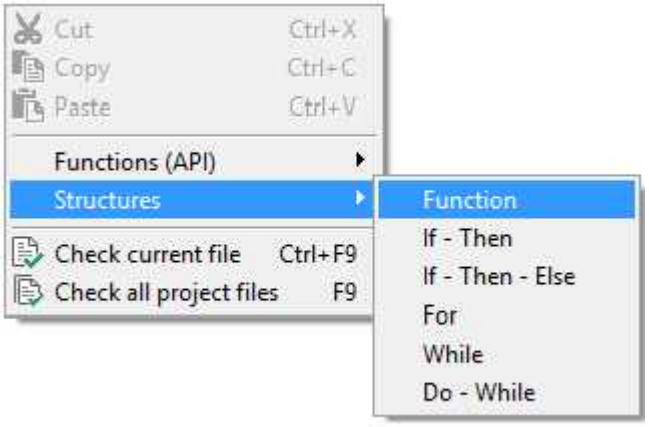
Right mouse buttons open a context menu with some common functions that aid user to write code



API let you choose all built-in functions in the libraries, you can browse them in menulike way. Click on the name and the CBuilder add prototype in the current code window at the cursor position.



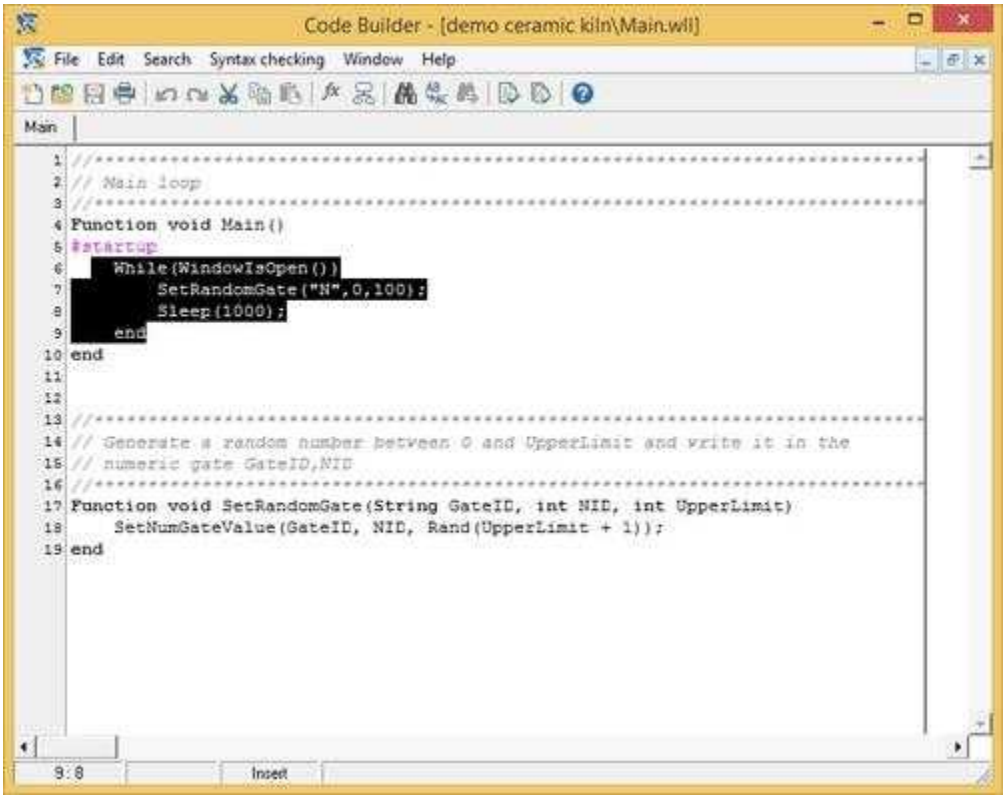
Structures like *functions* let you browse and add language structures.



6 The use of the clipboard

The clipboard is an area, that can't be accessed directly, where you can insert or remove text.

The text on which you want to do some deleting or transfer operations must be, first of all, selected. To select the text you have to hold down the SHIFT key and move the cursor on the screen so as to highlight the desired text. As an alternative, you can use the mouse holding down the left button.



All the highlighted text (in figure below) will be the object of the desired operation (chosen in the edit menu).

After having selected the desired text it is possible to:

Copy it in the clipboard
Cut it, that is to say delete it from the code and move it to the clipboard

To *paste* text from the clipboard you don't have to do any selection, the text in the clipboard will be inserted where the cursor is.

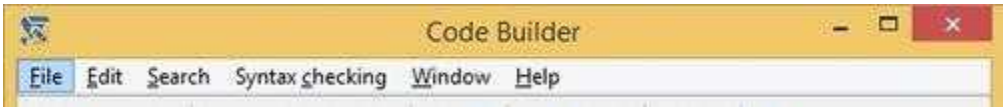
The speedbar button performing the *paste* function is colored (icon below) only when in the clipboard there is something that can be copied.

Similarly, the buttons for copying and deleting blocks of text, are highlighted only when, in the code you are working on, there is some selected text.

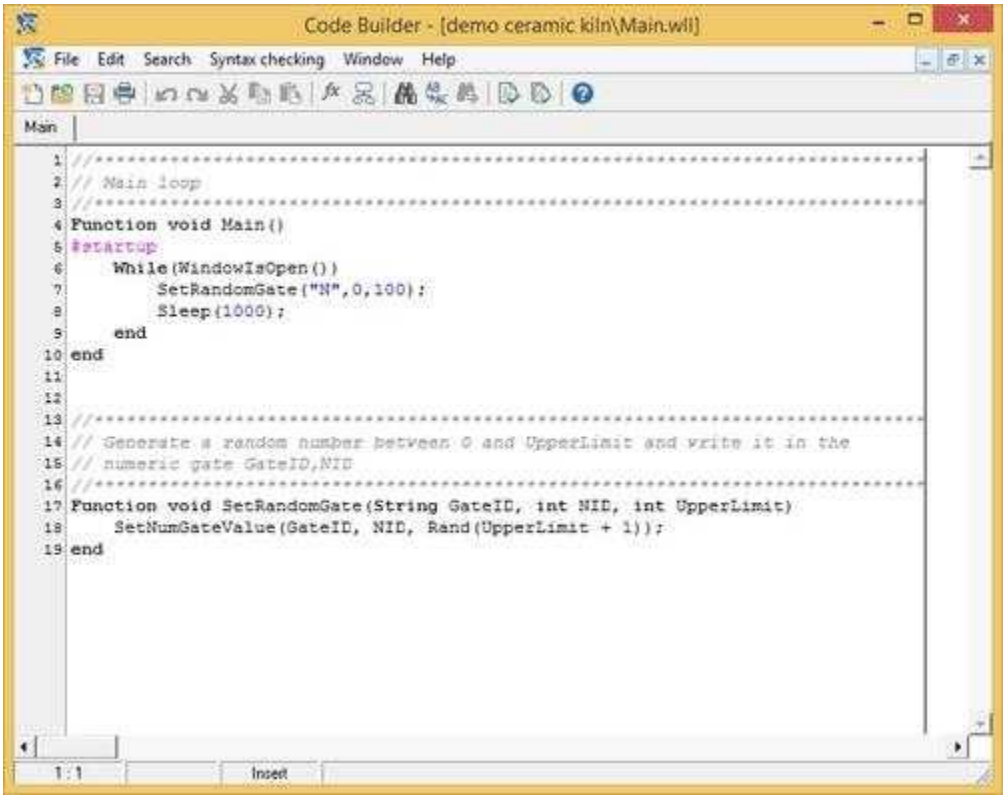
7 Menus

7.1 Introduction

Now we are going to see in detail the functions provided by the menus.



While you are working, there are two ways to access the menus: the first is to use the mouse, the second is to use the ALT key in combination with the underlined letter of the menu.

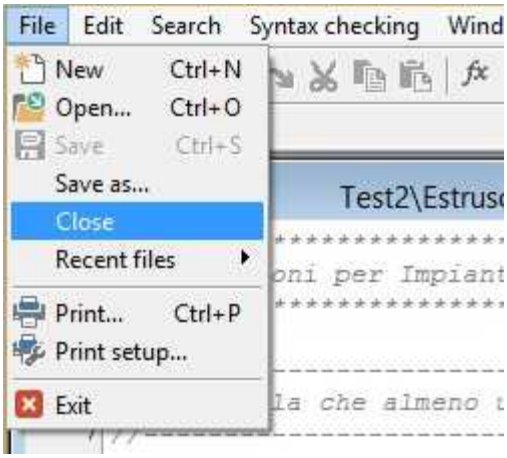


The functions provided by the menus can be applied to the current code window, that is to say the one that, compared to the others, is in the foreground (as shown in figure above).

7.2 Files

7.2.1 Introduction

The *File* menu includes all the functions necessary to permanently store and print the code.



From this menu you can open already existing files, save what you are working on, print it or close it. Beside every option of the menu you may note the indication of one or more keys; for example the option *new* corresponds to the combination CTRL+N.

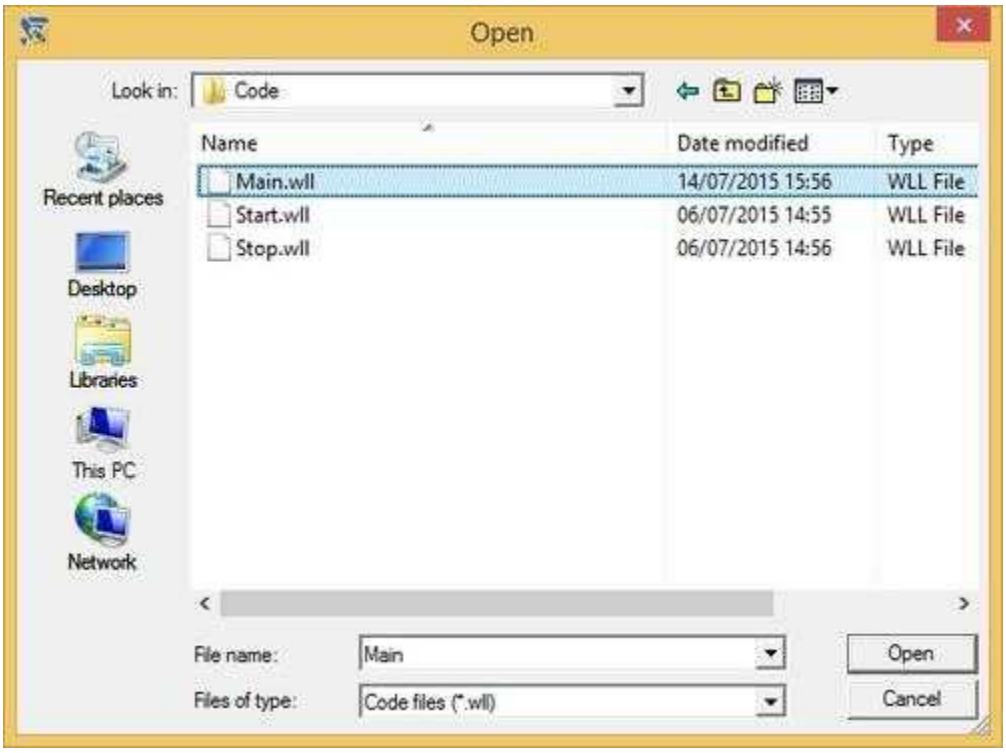
The three dots (...) following an item indicate that this opens a dialog box:

7.2.2 New

New: it opens an empty window in the workspace.

7.2.3 Open

Open: it shows a dialog box in which you can select the file you want to work on.
Once you have selected the desired file, just press the button *Open*.



7.2.4 Save

Save: it stores the file you are working on; the first time you save a file, created with the command *new,* you'll see a dialog box similar to the previous one, in which you can save the file with a name automatically created, or give a new name to it. The data will be automatically saved without using the dialog box, of course if the file has been saved at least once before.

7.2.5 Save As

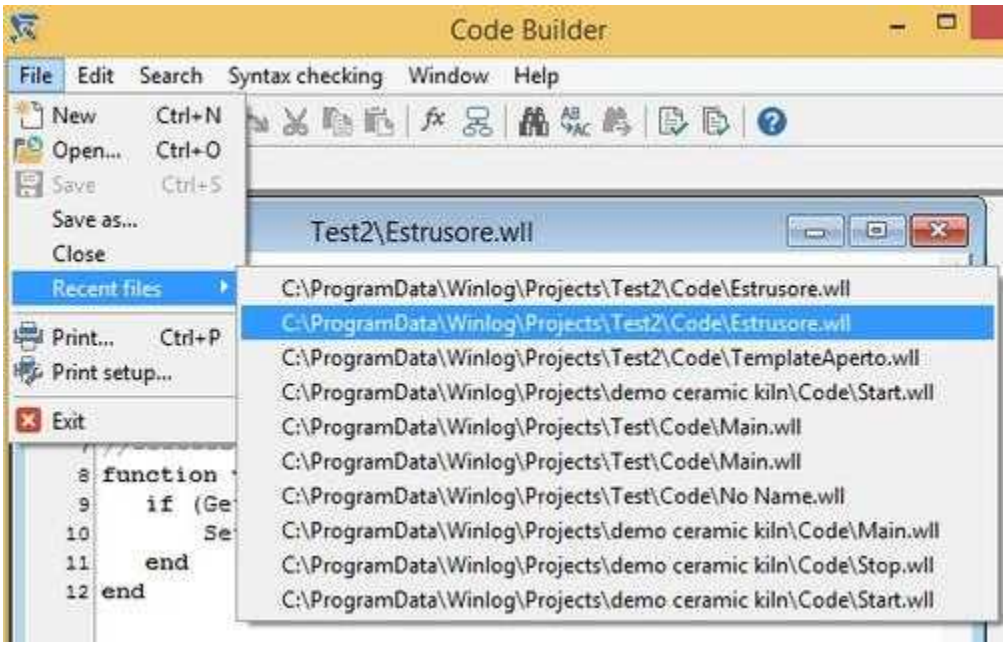
Save as: it opens every time the dialog (the same of Save) box for saving the file.

7.2.6 Close

Close: it closes the window code presently in the foreground.

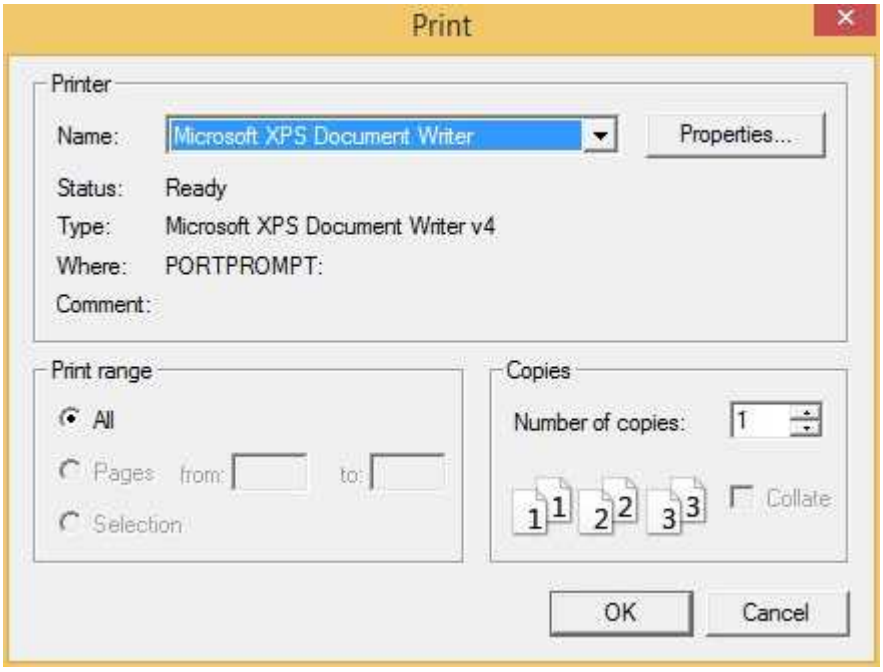
7.2.7 Recent Files

Recent files: show you a list of recent used files (max 10 item).



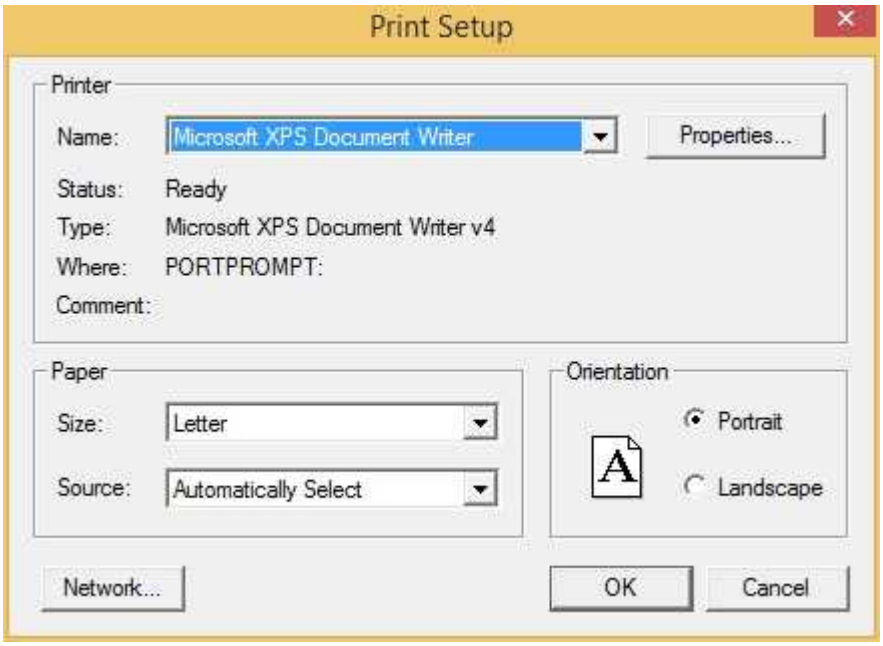
7.2.8 Print

Print: it sends to the printer the content of the window code you are working on.



7.2.9 Print Setup

Printer setup: it configures the print options.



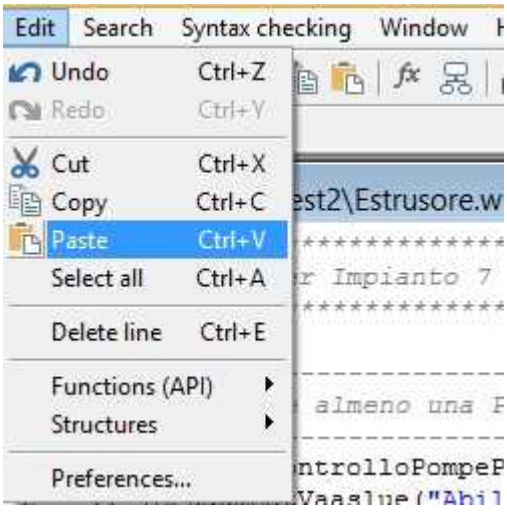
7.2.10 Exit

Exit: Close CBuilder and current work session.

7.3 Edit

7.3.1 Introduction

The *edit* menu includes all the functions necessary for the editing of text blocks, the correction of wrong operations and the configuration of the working environment.



7.3.2 Undo

Undo: it restores the last text deleted. This can be a character or a block of characters.

7.3.3 Redo

Redo: it restores the last 'Undo' operation

7.3.4 Cut

Cut: it deletes the text selected in the window code and copies it in a memory area (clipboard) from where it can be read later on.

7.3.5 Copy

Copy: it takes the selected block of characters and copies it in the clipboard.

7.3.6 Paste

Paste: it reads the content of the clipboard and inserts it in the code, where the cursor is at that moment.

7.3.7 Select all

Select all: Select all the text in the code file

7.3.8 Delete line

Delete line: Delete text in the current line without copying it in the clipboard.

7.3.9 Functions(API)

Functions(API): it allows to select and insert a function from the list of the available API

7.3.10 Structures

Structures: it allows to select and insert a structure from the list of the available structures

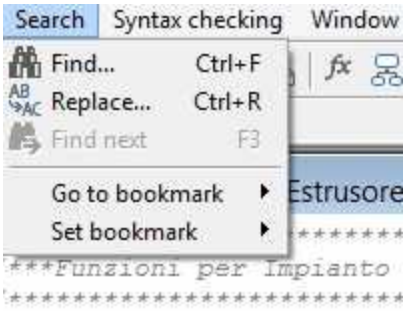
7.3.11 Preferences

Preferences: it opens a dialog box where it is possible to configure the environment at will.

7.4 Search

7.4.1 Introduction

In this menu there are the functions needed to search and replace text.



7.4.2 Find

Find: it opens a dialog box (figure below) where you can insert one or more words you want to search.

In this box you can specify the search parameters. Since the operation is carried out starting from the present position of the cursor, you can decide to move up (*Up*) or down (*Down*) the code. By ticking the box *Match case*, you can require that the word is searched exactly as it had been inserted. This way the search will distinguish capitals from small letters. After having inserted the text in the box, press the *Find next* button: the cursor will position itself on the first word or expression found in the text.

7.4.3 Replace

Replace: it opens a dialog box where you can specify the text to search and replace with a new one.

It is possible to specify if the search must distinguish capitals from small letters like in the previous function. The button *Replace All* repeats the inserted replacement for all the words or expressions found in the text.

7.4.4 Find next

Find next: it repeats the last search without opening any dialog box.

7.4.5 Go to bookmark

Go to bookmark: it allows to browse between bookmarks

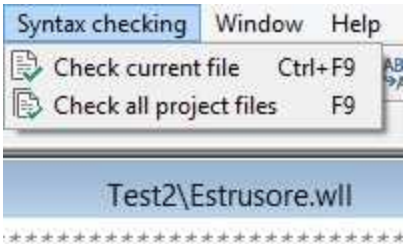
7.4.6 Set bookmark

Set bookmark: it allows to set a bookmark

7.5 Syntax checking

7.5.1 Introduction

This menu offers all the functions necessary for the syntactic control of the code.



7.5.2 Check current file

Check current file: it checks the file you are working on, searching for any syntactic mistake.

7.5.3 Check all project files

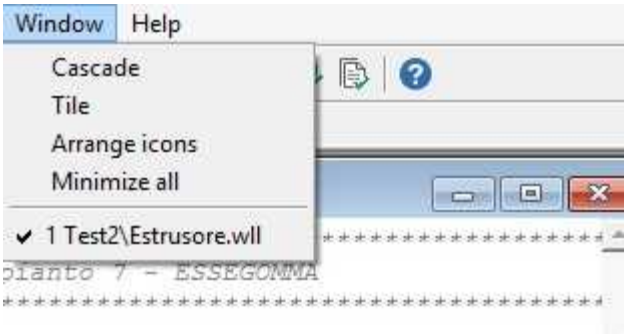
Check all project files: it checks all the project, in its entirety, searching for mistakes. If it finds one, the cursor will position itself where the mistake has been found.

7.6 Window

7.6.1 Introduction

In this menu there are the commands for the management of the windows in the workspace.

At the bottom of the menu you can see the list of the code windows present: to access one of them you just need to select it from the menu.



7.6.2 Cascade

Cascade: it resizes and positions all the windows in the workspace, putting one over another, but slightly moving them, so as to be able to access each of them.

7.6.3 Tile

Tile: it divides the workspace giving all the code windows the same space.

7.6.4 Arrange icons

Arrange icons: it arranges the icons of the minimized windows.

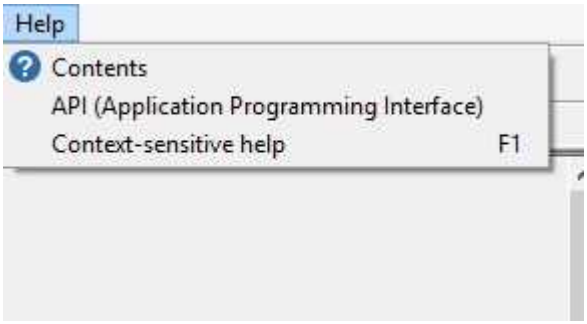
7.6.5 Minimize all

Minimize all: it reduces all the code windows to the minimum size and sort them.

7.7 Help

7.7.1 Introduction

If you need help it can be useful to read the guide or the contextual help, both accessible from the *Help* menu.



7.7.2 Contents

Contents: Show you the tree contents of the guide.

7.7.3 API (Application Programming Interface)

API (Application Programming Interface): Show you the guide of the API.

7.7.4 Context-sensitive help

Context-sensitive help: Help you starting from the introduction.

8 How to configure the integrated environment

8.1 Introduction

There are some aspects of the integrated environment that can be configured. You can do it by selecting *Preferences* from the *Edit* menu; a number of pages will appear: every page has a different name, *colors, editor*, etc.

To access a page different from the present one you have to click on the corresponding name. Now we are going to see in detail what you can configure in these pages.

8.2 Colors

It is possible to modify the color and the style of the text elements at will. The rules differentiating the text elements come directly from the language syntax.

By selecting an element from the *Elements* list (figure below), the page will be updated automatically, showing the configuration for that element.

For example, as shown in figure below, the element *Reserved words* has *bold* for style, white for background color and black for text color.

```
Function Void New_Recipe()
#Macro
  String name = InputDialog("Recipe name","
  if (name!="") then
      if (RecipeCreate("Production recipes"
         RecipeImport("Production recipes",
      end
   end
End
```

To change the style of an element you just need to tick or remove the tick from one of the boxes grouped in the item *styles*. Similarly, to choose the color you will have to click on one of the colored squares: by clicking with the mouse left button you select the text color, (FG) while, using the right button you define the background color (BG).

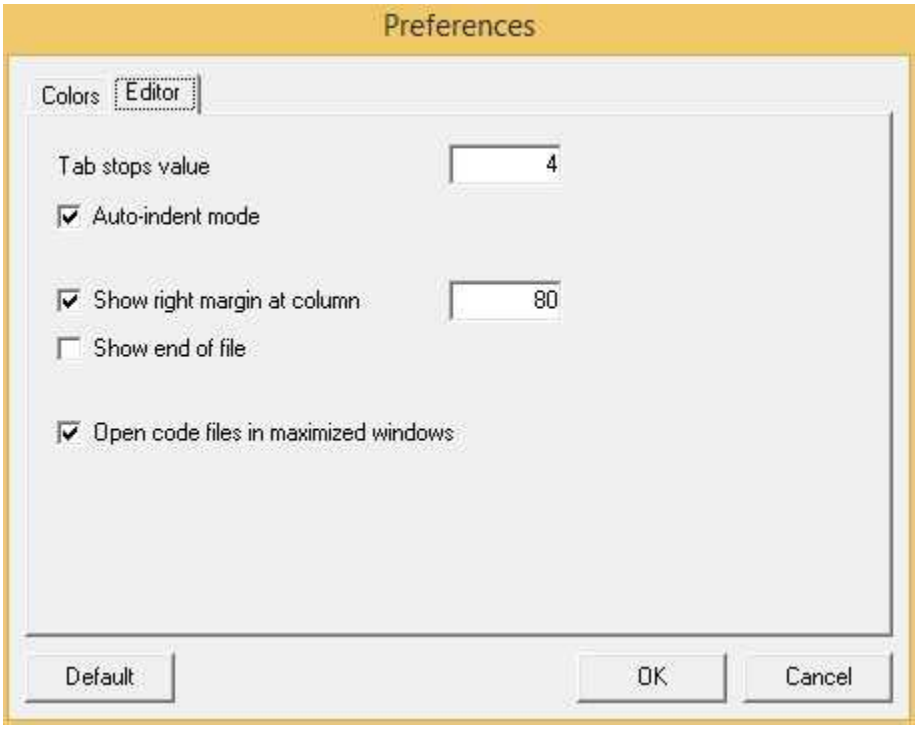
You can configure every element to your liking and confirm the changes made pressing the button *OK*; if you want to undo the changes just press the button *Cancel*.

To restore the default options you can press the button *Default.*

The item "*use default for"* indicates whether you can use for every element the selected background color or you have to use for all the items the background color defined by Windows.

8.3 Editor

In the Editor page (shown in figure) there are functions concerning the code windows.



Looking at a code window, you may note the presence of a vertical and two horizontal grey lines:

- The vertical line represents a limit that it is better not to overcome: you can choose if you want to display this limit, and also in which column to place it.
- The two horizontal lines signal, in the code window, the end of the file. They also can be omitted.

Pressing the TAB key in a code window, the cursor moves as far as the next tabulation: the tabulations size can be defined through the item *tabulations width*.

Persistent Block is the way you can avoid deletion of current selected block while pasting data. Click its check box so selected blocks won't be cancelled if you paste with CTRL+V or menu subfunction (see cut&paste)

9 API (Application Programming Interface)

9.1 Introduction

With API is intended to indicate the application programming interface that is the set of all instructions available to the developer to meet the various needs of the application, such as create or read text files, automatically change gates value to the occurrence of particular events or conditions, send email or SMS, creating reports of production, import, or save recipes, work with objects on the templates, perform mathematical calculations, and so on.

The instructions in this guide are available under the category *API* grouped according to the subject to which they refer: the category "Files" for example, contains all the instructions that deal with the management of the file, while the category "Gates" all instructions operating on gates.

Example

```
Function Void Test()
      real v = GetNumGateValue("N",1);
      v = v + 1;
      SetNumGateValue("N",1,v);
end
```

In the function above, GetNumGateValue () and SetNumGateValue () are two instructions *API* of *Gates* category.
In the Code Builder, by placing the cursor on it and pressing the F1 key you can automatically call up the detailed guide.

9.2 Bit

9.2.1 BitAnd

Description
Makes the bitwise AND between Value1 and Value2

Syntax
```
int BitAnd(int Value1, int Value2)
```

Parameters
Value1 first number to process
Value2 second number to process

Returned value
The result of operation Value1 **AND** Value2

Related functions
BitOr(),BitXor(),BitNot()

Example
```
int Result;
int Value1;
int Value2;
Value1=HexStrToInt("0xFFA7");
Value2=HexStrToInt("0x0FF5");
Result=BitAnd(Value1,Value2);
```

The Result is 0xFA5

9.2.2 BitMask

Description
it makes the bitwise AND between a number and a mask

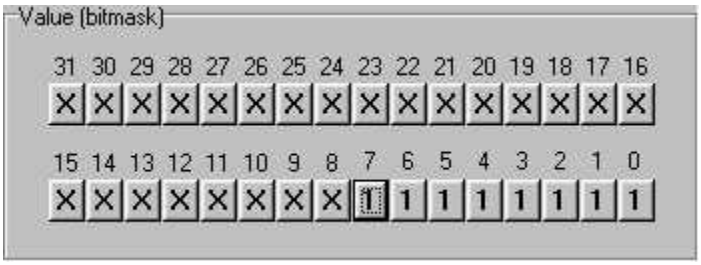
Syntax
```
int BitMask(int Num, int Mask)
```

Parameters
Num number to process
Mask mask (bit position from 0 to 31)

Returned value
the result of operation

Related functions
SetBit(), ResetBit(), GetBit()

Example
```
LowByte=BitMask(Value,HexStrToInt("000000FF"));
```



9.2.3 BitNot

Description
Makes the bit negation of Value

Syntax
```
int BitNot(int Value)
```

Parameters
Value value to process

Returned value
The result of operation **NOT** Value

Related functions
BitOr(),BitXor(),BitAnd()

Example
```
int Result;
int Value;
Value=HexStrToInt("0xFFA7");
Result=BitNot(Value);
```

The Result is 0x0058

9.2.4 BitOr

Description
Makes the bitwise OR between Value1 and Value2

Syntax
```
int BitOr(int Value1, int Value2)
```

Parameters
Value1 first number to process
Value2 second number to process

Returned value
The result of operation Value1 **OR** Value2

Related functions
BitXor(),BitAnd(),BitNot()

Example
```
int Result;
int Value1;
int Value2;
Value1=HexStrToInt("0xFFA7");
Value2=HexStrToInt("0x0FF5");
Result=BitOr(Value1,Value2);
```

The Result is 0xFFF7

9.2.5 BitXor

Description
Makes the bitwise XOR between Value1 and Value2

Syntax
```
int BitXor(int Value1, int Value2)
```

Parameters
Value1 first number to process
Value2 second number to process

Returned value
The result of operation Value1 **XOR** Value2

Related functions
BitOr(),BitAnd(),BitNot()

Example
```
int Result;
int Value1;
int Value2;
Value1=HexStrToInt("0xFFA7");
Value2=HexStrToInt("0x0FF5");
Result=BitXor(Value1,Value2);
```

The Result is 0xF052

9.2.6 GetBit

Description
it returns you a bit from a number

Syntax
```
int GetBit(int Num, int Bit)
```

Parameters
Num number to check
Bit Bit to check (position from 0 to 31)

Returned value
requeste bit

Related functions
BitMask(), SetBit(),ResetBit()

Example
```
Even =GetBit(A,0);
```

9.2.7 ResetBit

Description
it resets (set to 0) a specified bit of a number

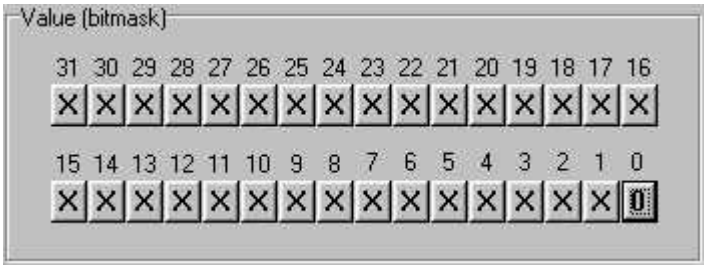
Syntax
```
int ResetBit(int Num, int Bit)
```

Parameters
Num number to process
Bit Bit you want to reset (bit position from 0 to 31)

Returned value
number with the bit low.

Related functions
BitMask(), SetBit(), GetBit()

Example
```
A_even=ResetBit(A,0);
```



9.2.8 SetBit

Description
it sets (to 1) a specified bit of a number

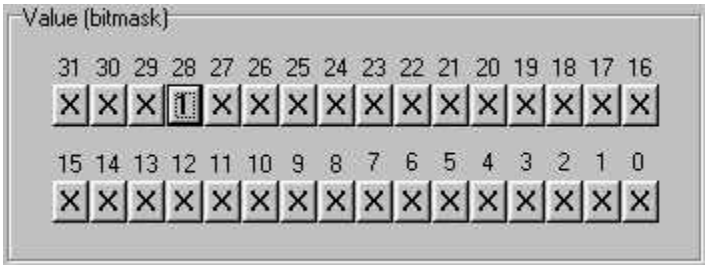
Syntax
```
int SetBit(int Num, int Bit)
```

Parameters
Num integer to process
Bit Bit you want to set (bit position from 0 to 31)

Returned value
the number with the bit high

Related functions
BitMask(), ResetBit(), GetBit()

Example
```
SetBit(0,28);
```



9.3 Date and Time

9.3.1 DateTimeToSeconds

Description
Converts date time received in input to total seconds elapsed since January 1, 1901.

Syntax
```
Unsigned DateTimeToSeconds(
                           int Day,int Month,int Year,int Hour,int
                           Minute,int Second)
```

Parameters

Day	day of the date/time to convert in seconds since January 1, 1901.
Month	month of the date/time to convert in seconds since January 1, 1901.
Year	Year of the date/time to convert in seconds since January 1, 1901.
Hour	hour of the date/time to convert in seconds since January 1, 1901.
Minute	minute of the date/time to convert in seconds since January 1, 1901.
Seconds	second of the date/time to convert in seconds since January 1, 1901.

Returned value
the number of seconds elapsed since January 1, 1901.

Related functions
GetDayFromSeconds(),GetMonthFromSeconds(),GetYearFromSeconds(), GetHourFromSeconds(), GetMinuteFromSeconds(),GetSecondFromSeconds()

Example
```
Unsigned Seconds;
Seconds=DateTimeToSeconds(10,7,1970,5,34,22);
```

9.3.2 GetDateString

Description
It returns a string containing the date. In it days, months and years are separated by a specified punctuation character; it is possible to specify whether you want the date to be written according to the European style.

Syntax
```
String GetDateString(String Separator, Bool EuropeanStyle)
```

Parameters

Separator	the character separating days, months and years
EuropeanStyle	it indicates if you want to write the date in European style

Returned value
the date written as a string with the required specifications

Related functions
GetYear(), GetMonth(), GetDayOfMonths(), GetDayOfWeek()

Example
```
CurrentDate=GetDateString("/",true);
```

9.3.3 GetDayFromSeconds

Description
Returns the day of the date/time corresponding to the total seconds since January 1, 1901 passed in input to the instruction.

Syntax
```
Int GetDayFromSeconds(Unsigned Seconds)
```

Parameters

Seconds	total amount of seconds since January 1, 1901.

Returned value
the number of the day corresponding.

Related functions
DateTimeToSeconds(),GetMonthFromSeconds(),GetYearFromSeconds(), GetHourFromSeconds(), GetMinuteFromSeconds(),GetSecondFromSeconds()

Example
```
Unsigned Seconds;
Int Day;
Seconds=DateTimeToSeconds(10,7,1970,5,34,22);
Seconds=Seconds+90000;        // Add 25 hours
Day=GetDayFromSeconds(Seconds);
```

9.3.4 GetDayOfMonth

Description
It returns an integer number representing the current day.

Syntax
```
Int GetDayOfMonth()
```

Parameters
-

Returned value
the current day as an integer number

Related functions
GetDateString(), GetYear(),GetMonth(), GetDayOfWeek()

Example

```
TodayIs=GetDayOfMonth();
```

9.3.5 GetDayOfWeek

Description
It returns an integer number representing the current day of the week (0=Sunday, 1 =Monday, 2=Tuesday, ...).

Syntax
```
Int GetDayOfWeek()
```

Parameters
-

Returned value
the current day of the week as an integer number

Related functions
GetDateString(), GetYear(), GetDayOfMonth(), GetMonth()

Example
```
if ( GetDayOfWeek()==0 ) then
      Sunday=true;
end
```

9.3.6 GetHour

Description
It returns an integer number representing the current hour; minutes, seconds and milliseconds are not in the returned value.

Syntax
```
Int GetHour()
```

Parameters
-

Returned value
the hour as an integer number

Related functions
GetTimeString(), GetMinute(), GetSecond(), GetMilliSeconds()

Example
```
Hour=GetHour();
```

9.3.7 GetHourFromSeconds

Description
Returns the hour of the date/time corresponding to the total seconds since January 1, 1901 passed in input to the instruction.

Syntax
```
Int GetHourFromSeconds(Unsigned Seconds)
```

Parameters
Seconds total amount of seconds since January 1, 1901.

Returned value
the number of the hour corresponding.

Related functions
DateTimeToSeconds(),GetDayFromSeconds(),GetMonthFromSeconds(),GetYearFromSeconds(), GetMinuteFromSeconds(),GetSecondFromSeconds()

Example
```
Unsigned Seconds;
Int Hour
Seconds=DateTimeToSeconds(10,7,1970,5,34,22);
Seconds=Seconds+90000;        // Add 25 hours
Hour=GetHourFromSeconds(Seconds);
```

9.3.8 GetMilliseconds

Description
It returns an integer number representing the number of milliseconds of the current time.

Syntax
```
Int GetMilliseconds()
```

Parameters
-

Returned value
the number of milliseconds of the current time as an integer number

Related functions
GetTimeString(), GetHour(), GetMinute(), GetSecond()

Example
```
MSecs=GetSecond() / 60 + GetMilliSeconds();
```

9.3.9 GetMinute

Description
It returns an integer number representing the number of minutes of the current hour.

Syntax
```
Int GetMinute()
```

Parameters
-

Returned value
the number of minutes of the current hour as an integer number

Related functions
GetTimeString(), GetHour(), GetSecond(), GetMilliSeconds()

Example
```
Secs=GetMinute()*60;
```

9.3.10 GetMinuteFromSeconds

Description
Returns the minute of the date/time corresponding to the total seconds since January 1, 1901 passed

in input to the instruction.

Syntax
```
Int GetMinuteFromSeconds(Unsigned Seconds)
```

Parameters
Seconds total amount of seconds since January 1, 1901.

Returned value
the number of the minute corresponding.

Related functions
DateTimeToSeconds(),GetDayFromSeconds(),GetMonthFromSeconds(),GetYearFromSeconds(), GetHourFromSeconds(),GetSecondFromSeconds()

Example
```
Unsigned Seconds;
Int Minute
Seconds=DateTimeToSeconds(10,7,1970,5,34,22);
Seconds=Seconds+90000;        // Add 25 hours
Minute=GetMinuteFromSeconds(Seconds);
```

9.3.11 GetMonth

Description
It returns an integer number representing the current month.

Syntax
```
Int GetMonth()
```

Parameters
-

Returned value
the current month as an integer number

Related functions
GetDateString(), GetYear(), GetDayOfMonths(), GetDayOfWeek()

Example
```
MonthLeft=12-GetMonth();
```

9.3.12 GetMonthFromSeconds

Description
Returns the month of the date/time corresponding to the total seconds since January 1, 1901 passed in input to the instruction.

Syntax
```
Int GetMonthFromSeconds(Unsigned Seconds)
```

Parameters
Seconds total amount of seconds since January 1, 1901.

Returned value
the number of the month corresponding.

Related functions

DateTimeToSeconds(),GetDayFromSeconds(),GetYearFromSeconds(), GetHourFromSeconds(), GetMinuteFromSeconds(),GetSecondFromSeconds()

Example
```
Unsigned Seconds;
Int Month;
Seconds=DateTimeToSeconds(10,7,1970,5,34,22);
Seconds=Seconds+90000;        // Add 25 hours
Month=GetMonthFromSeconds(Seconds);
```

9.3.13 GetSecond

Description
It returns an integer number representing the number of the seconds of the current time.

Syntax
```
Int GetSecond()
```

Parameters
-

Returned value
the number of seconds of the current time as an integer number

Related functions
GetTimeString(), GetHour(), GetMinute(), GetMilliSeconds()

Example
```
Secs=GetMinute()*60 + GetSecond();
```

9.3.14 GetSecondFromSeconds

Description
Returns the second of the date/time corresponding to the total seconds since January 1, 1901 passed in input to the instruction.

Syntax
```
Int GetSecondFromSeconds(Unsigned Seconds)
```

Parameters
Seconds total amount of seconds since January 1, 1901.

Returned value
the number of the second corresponding.

Related functions
DateTimeToSeconds(),GetDayFromSeconds(),GetMonthFromSeconds(),GetYearFromSeconds(), GetHourFromSeconds(),GetMinuteFromSeconds()

Example
```
Unsigned Seconds;
Int Second
Seconds=DateTimeToSeconds(10,7,1970,5,34,22);
Seconds=Seconds+90000;        // Add 25 hours
Second=GetSecondFromSeconds(Seconds);
```

9.3.15 GetTickCount

(OBSOLETE)

Description
It returns an integer number representing the number of milliseconds elapsed since the start of the system. After about 24.8 days this number is 0 again.

Syntax
```
Int GetTickCount()
```

Parameters
-

Returned value
the number of milliseconds elapsed since the start of the system as an integer number

Related functions
-

Example
```
TimeFromStart = GetTickCount() / TickSize;
```

Note: It's recommended to use UnsignedGetTickCount().

9.3.16 GetTimeString

Description
It returns a string containing the time. In it hour, minutes, seconds and milliseconds are separated by a specified punctuation character.

Syntax
```
String GetTimeString(String Separator)
```

Parameters
Separator the character separating hour, minutes and seconds

Returned value
the time written as a string with the required separator

Related functions
GetHour(), GetMinute(), GetSecond(), GetMilliSeconds()

Example
```
CurrentTime=GetTimeString(":");
```

9.3.17 GetYear

Description
It returns an integer number representing the current year.

Syntax
Int GetYear()

Parameters

-

Returned value
the current year as an integer number

Related functions
GetDateString(), GetMonth(), GetDayOfMonths(), GetDayOfWeek()

Example
```
Deltayear=GetYear()-1980;
```

9.3.18 GetYearFromSeconds

Description
Returns the year of the date/time corresponding to the total seconds since January 1, 1901 passed in input to the instruction.

Syntax
```
Int GetYearFromSeconds(Unsigned Seconds)
```

Parameters
Seconds total amount of seconds since January 1, 1901.

Returned value
the number of the year corresponding.

Related functions
DateTimeToSeconds(),GetDayFromSeconds(),GetMonthFromSeconds(),GetHourFromSeconds(), GetMinuteFromSeconds(),GetSecondFromSeconds()

Example
```
Unsigned Seconds;
Int Year
Seconds=DateTimeToSeconds(10,7,1970,5,34,22);
Seconds=Seconds+90000;         // Add 25 hours
Year=GetYearFromSeconds(Seconds);
```

9.3.19 SetDateTime

Description
Set Date and Time values.
Runtime must be executed with Administrator rights to let SetDateTime() to take effect.

Syntax
Bool SetDateTime(Int Day,int Month,int Year,int Hour,int Minutes,int Seconds)

Parameters

Day	day to set
Month	month to set
Year	year to set
Hour	hour to set
Minutes	minutes to set
Seconds	seconds to set

Returned value
True : if function terminated correctly.

False: if an error happens.

Related functions
-

Example
```
SetDateTime(10,7,1970,10,11,50);
```

9.3.20 UnsignedGetTickCount

Description
It returns an unsigned integer number representing the number of milliseconds elapsed since the start of the system. After about 49.7 days this number is 0 again.

Syntax
```
Unsigned UnsignedGetTickCount()
```

Parameters
-

Returned value
the number of milliseconds elapsed since the start of the system as an unsignedt integer number

Related functions
-

Example
```
Unsigned TimeFromStart;
TimeFromStart = UnsignedGetTickCount();
```

9.4 Devices

9.4.1 DeviceEnableCommunication

Description
Enable or disable communication with the specified device.

Syntax
```
Bool DeviceEnableCommunication(
                                  Int Channel, Int DevNum,Bool ToEnable,Bool
                                  SaveChanges)
```

Parameters
Channel channel number
DevNum device number
ToEnable
if "true" the communication with device will be enabled.
if "false" the communication with device will be disabled.
SaveChanges
if "true" the changes will be permanently saved.
if "false" the changes will be valid only in the present session.

Returned value
"true" if function has been executed correctly,
"false" on error.

Related functions
IsDeviceCommunicationEnabled()

Example
// Enable communication with device 3 of channel 1
```
DeviceEnableCommunication(1,3,true);
```

9.4.2 DeviceName

Description
Returns the name of the device specified by channel number and device name.

Syntax
```
String DeviceName(Int Channel, Int DevNum)
```

Parameters
Channel channel number
DevNum device number

Returned value
name of the device

Related functions
-

Example
```
DevName = DeviceName (1,3);
```

9.4.3 GetDeviceRxErrors

Description
Return the number of communication errors happened douring all data read operations from the specified device.

Syntax
```
Int GetDeviceRxErrors(Int Channel, Int DevNum)
```

Parameters
Channel channel number
DevNum device number

Returned value
Number of communication errors.
"-1" on istruction failure.

Related functions
ResetDeviceRxErrors(),GetDeviceTxErrors()

Example
Int RxErrors;
```
RxErrors=GetDeviceRxErrors(1,3);
```

9.4.4 GetDeviceTxErrors

Description
Return the number of communication errors happened douring all data write operations towards the specified device.

Syntax
```
Int GetDeviceTxErrors(Int Channel, Int DevNum)
```

Parameters
Channel channel number
DevNum device number

Returned value
Number of communication errors.
"-1" on istruction failure.

Related functions
ResetDeviceTxErrors(),GetDeviceRxErrors()

Example
```
Int TxErrors;
TxErrors=GetDeviceTxErrors(1,3);
```

9.4.5 IsDeviceCommunicationEnabled

Description
Return the enabled/disabled device communication status.

Syntax
```
Bool IsDeviceCommunicationEnabled(Int Channel, Int DevNum)
```

Parameters
Channel channel number
DevNum device number

Returned value
"true" if device communication is enabled.
"false" if device communication is disabled.

Related functions
DeviceEnableCommunication()

Example
```
Bool Status;
Status=IsDeviceCommunicationEnabled(1,3);
```

9.4.6 IsDeviceCommunicationKo

Description
Return the device communication status (Ok/Ko).

Syntax
```
Bool IsDeviceCommunicationKo(Int Channel, Int DevNum)
```

Parameters
Channel channel number
DevNum device number

Returned value
"true" if the communication with the device is KO
"false" if the communication with the device is OK

Related functions
IsDeviceCommunicationEnabled()

Example
```
Bool ComKo;
ComKo=IsDeviceCommunicationKo(1,3);
```

9.4.7 ResetDeviceRxErrors

Description
Reset read errors communication counter of the device.

Syntax
```
Bool ResetDeviceRxErrors(Int Channel, Int DevNum)
```

Parameters
Channel	channel number
DevNum	device number

Returned value
"true" if function has been executed correctly,
"false" on error.

Related functions
GetDeviceRxErrors(),ResetDeviceTxErrors()

Example
```
ResetDeviceRxErrors(1,3);
```

9.4.8 ResetDeviceTxErrors

Description
Reset write errors communication counter of the device.

Syntax
```
Bool ResetDeviceTxErrors(Int Channel, Int DevNum)
```

Parameters
Channel	channel number
DevNum	device number

Returned value
"true" if function has been executed correctly,
"false" on error.

Related functions
GetDeviceTxErrors(),ResetDeviceRxErrors()

Example
```
ResetDeviceTxErrors(1,3);
```

9.5 Directory

9.5.1 DirectoryCreate

Description
it creates a new directory

Syntax

Bool DirectoryCreate(String DirName)

Parameters
DirName name of new directory

Returned value
true (directory successfully created)
false (on error)

Related functions
DirectoryDelete()

Example
```
DirectoryCreate(GetProjectPath() + "\MyDirectory");
```

9.5.2 DirectoryDelete

Description
it deletes a directory; if directory is not empty it will not be deleted.

Syntax
Bool DirectoryDelete(String DirName)

Parameters
DirName name of the directory you want to delete

Returned value
true (directory deleted)
false (on failure)

Related functions
DirectoryCreate()

Example
```
DirectoryDelete(GetProjectPath() + "\MyDirectory");
```

9.5.3 DirectoryGetCurrent

Description
it returns the current directory

Syntax
String DirectoryGetCurrent()

Parameters
-

Returned value
the current path

Related functions
DirectorySetCurrent()

Example
```
SearchFor=DirectoryGetCurrent()+"\*.*";
```

9.5.4 DirectorySetCurrent

Description
it sets the current directory

Syntax
Bool DirectorySetCurrent(String NewPath)

Parameters
NewPath new path you want to set

Returned value
true (current path updated)
false (on error)

Related functions
DirectoryGetCurrent()

Example
```
DirectorySetCurrent("y:\projects");
```

9.6 Files

9.6.1 FileAttrFound

(OBSOLETE)
It's recommended to use FileAttrFoundEx().

Description
It returns the file attribute of last file read by *FileFindFirst* or *FileFindNext*

Syntax
```
Int FileAttrFound()
```

Parameters
-

Returned value
attribute of last file found

Related functions
FileFindNext(), FileFindClose(), FileFindFirst(), FileNameFound()

Example
```
Function void ExampleFindFirst()
        String CurrPath;
        String CurrFile;
        CurrPath=DirectoryGetCurrent()+"\*.*";
        if (FileFindFirst(CurrPath)==0) then
                CurrFile=FileNameFound();
                MessageBox(CurrFile,CurrPath);
                while (FileFindNext() == 0)
                        CurrFile=FileNameFound();
                        MessageBox(CurrFile,CurrPath);
                end
        end
        FileFindClose();
End
```

9.6.2 FileAttrFoundEx

Description
It returns the file attribute of last file read by *FileFindFirstEx* or *FileFindNextEx*

Syntax
```
Int FileAttrFoundEx(int Handle)
```

Parameters
Handle this is the handle of the search in progress.

Returned value
attribute of last file found

Related functions
FileFindFirstEx() , FileFindNextEx() , FileFindCloseEx() , FileNameFoundEx()

Example
```
Function void FindFile()
	String CurrPath;
	String CurrFile;
	int Handle;
	CurrPath=GetProjectPath()+"\CSV\*.csv";
	Handle=FileFindFirstEx(CurrPath);
	if (Handle!=0) then
		CurrFile=FileNameFoundEx(Handle);
		if (CurrFile!="") then
			MessageBox("File Foud: "+CurrFile,"FileNameFoudEx");
			while (FileFindNextEx(Handle)!=false)
				CurrFile=FileNameFoundEx(Handle);
				MessageBox("File Foud: "+CurrFile,"FileNameFoudEx");
			end
		end
		FileFindCloseEx(Handle);
	end
end
```

9.6.3 FileCopy

Description
Copies a file

Syntax
```
Bool FileCopy(String Src, String Dst, Bool IfExists)
```

Parameters

Src	file you want to copy
Dst	destination file
IfExists	**true** (if Dst exists it is not overwritten) **false** (if Dst exists it's overwritten)

Returned value
true (on success)
false (on failure)

Related functions
FileMove()

Example

```
if (FileExist("C:\Temporan\Promem.txt"))then
      FileCopy("C:\Temporan\Promem.txt","C:\Temporan\Promem.bak",false); //
overwrite if exists
end
```

9.6.4 FileClose

Description
Closes an already open file

Syntax
```
int FileClose(Int Handle)
```

Parameters
Handle handle of the file

Returned value
0 on success
non zero (on failure)

Related functions
FileOpen()

Example
```
FileHandle=FileOpen("C:\Temporan\Starter.dat","rt"); // open text file for
reading
FileWriteLn(FileHandle,"Stringa di prova");
FileWriteLn(FileHandle,"questa è la seconda riga");
FileClose(FileHandle);
```

9.6.5 FileDelete

Description
it deletes a file

Syntax
```
Bool FileDelete(String FileName)
```

Parameters
FileName name of the file

Returned value
true (on success)
false (on error)

Related functions
-

Example
```
if (FileExist("C:\Temporan\Promem.bak"))then
      FileDelete("C:\Temporan\Promem.bak");
end
```

9.6.6 FileEof

Description
it returns a nonzero value if reading/writing file pointer has arrived at the End Of File.

Syntax
```
Int FileEOF(Int Handle)
```

Parameters
Handle handle of opened file

Returned value
nonzero value at the end of the file

Related functions
-

Example
```
int i=0;
FileHandle=FileOpen("C:\Temporan\MadeByAPI.dat","rb"); // it opens binary
file for reading

While (FileEOF(FileHandle) == 0) // it read 3 chars at time
  i=i+1;
  s=FileRead(FileHandle,3);
  if (FileEOF(FileHandle) == 0) then// is data valid?
       MessageBox(s,"triplet number "+IntToStr(i));
  end
End
FileClose(FileHandle);
```

9.6.7 FileExist

Description
it tells you if the file exist

Syntax
```
Bool FileExist(String FileName)
```

Parameters
FileName name of the file

Returned value
true (it exists)
false (it does not exist)

Related functions
-

Example
```
if (FileExist("C:\Temporan\Promem.bak"))  then
      FileDelete("C:\Temporan\Promem.bak");
end
```

9.6.8 FileFindClose

(OBSOLETE)
It's recommended to use FileFindCloseEx().

Description
it stops the process of directory file reading (started with *FileFindFirst*) then free used resources

Syntax
`void FileFindClose()`

Parameters
-

Returned value
-

Related functions
FileFindNext(), FileFindFirst(), FileNameFound(), FileAttrFound()

Example
```
Function void ExampleFindFirst()
	String CurrPath;
	String CurrFile;
	CurrPath=DirectoryGetCurrent()+"\*.*";
	if (FileFindFirst(CurrPath)==0) then
		CurrFile=FileNameFound();
		MessageBox(CurrFile,CurrPath);
		while (FileFindNext() == 0)
			CurrFile=FileNameFound();
			MessageBox(CurrFile,CurrPath);
		end
	end
	FileFindClose();
End
```

9.6.9 FileFindCloseEx

Description
it stops the process of directory file reading (started with *FileFindFirstEx*) then free used resources

Syntax
`void FileFindCloseEx(int Handle)`

Parameters
Handle this is the handle of the search in progress.

Returned value
-

Related functions
FileFindFirstEx() , FileFindNextEx() , FileNameFoundEx() , FileAttrFoundEx()

Example
```
Function void FindFile()
	String CurrPath;
	String CurrFile;
	int Handle;
	CurrPath=GetProjectPath()+"\CSV\*.csv";
	Handle=FileFindFirstEx(CurrPath);
```

```
        if (Handle!=0) then
                CurrFile=FileNameFoundEx(Handle);
                if (CurrFile!="") then
                        MessageBox("File Foud: "+CurrFile,"FileNameFoudEx");
                        while (FileFindNextEx(Handle)!=false)
                                CurrFile=FileNameFoundEx(Handle);
                                MessageBox("File Foud: "+CurrFile,"FileNameFoudEx");
                        end
                end
                FileFindCloseEx(Handle);
        end
end
```

9.6.10 FileFindFirst

(OBSOLETE)
It's recommended to use FileFindFirstEx().

Description
it starts directory file reading process

Syntax
`Bool FileFindFirst(String Path)`

Parameters
Path path that you want to get directory file list from

Returned value
true (if exist at least a file)
false (empty directory)

Related functions
FileFindNext(), FileFindClose(), FileNameFound(), FileAttrFound()

Note: Don't execute more than one *FINDFIRST*, if you start using FindFirst you have to close the reading session with *FINDCLOSE* to free resources.

Example

```
Function void ExampleFindFirst()
        String CurrPath;
        String CurrFile;
        CurrPath=DirectoryGetCurrent()+"\*.*";
        if (FileFindFirst(CurrPath)==true) then
                CurrFile=FileNameFound();
                MessageBox(CurrFile,CurrPath);
                while (FileFindNext() == 0)
                        CurrFile=FileNameFound();
                        MessageBox(CurrFile,CurrPath);
                end
        end
        FileFindClose();
End
```

9.6.11 FileFindFirstEx

Description
it starts directory file reading process

Syntax
`int FileFindFirstEx(String Path)`

Parameters
Path path that you want to get directory file list from

Returned value
The handle for the search in progress. It must be used in the following instructions FileFindNextEx() and FileFindCloseEx().
0 means error.

To know if at last one file have been found check that FileNameFoundEx() returns a string different from "".

Related functions
FileFindNextEx() , FileFindCloseEx() , FileNameFoundEx() , FileAttrFoundEx()

Example
```
Function void FindFile()
	String CurrPath;
	String CurrFile;
	int Handle;
	CurrPath=GetProjectPath()+"\CSV\*.csv";
	Handle=FileFindFirstEx(CurrPath);
	if (Handle!=0) then
		CurrFile=FileNameFoundEx(Handle);
		if (CurrFile!="") then
			MessageBox("File Foud: "+CurrFile,"FileNameFoudEx");
			while (FileFindNextEx(Handle)!=false)
				CurrFile=FileNameFoundEx(Handle);
				MessageBox("File Foud: "+CurrFile,"FileNameFoudEx");
			end
		end
		FileFindCloseEx(Handle);
	end
end
```

9.6.12 FileFindNext

(OBSOLETE)

It's recommended to use FileFindNextEx().

Description
it goes on with the directory reading file process

Syntax
```
Bool FileFindNext()
```

Parameters
-

Returned value
true (another file found)
false (no more files)

Related functions
FileFindFirst(), FileFindClose(), FileNameFound(), FileAttrFound()

Example
```
Function void ExampleFindFirst()
	String CurrPath;
	String CurrFile;
	CurrPath=DirectoryGetCurrent()+"\*.*";
```

```
        if (FileFindFirst(CurrPath)==0) then
                CurrFile=FileNameFound();
                MessageBox(CurrFile,CurrPath);
                while (FileFindNext() == 0)
                        CurrFile=FileNameFound();
                        MessageBox(CurrFile,CurrPath);
                end
        end
        FileFindClose();
End
```

9.6.13 FileFindNextEx

Description
it goes on with the directory reading file process

Syntax
`bool FileFindNextEx(int Handle)`

Parameters
Handle this is the handle of the search in progress.

Returned value
true (another file found)
false (no more files)

Related functions
FileFindFirstEx() , FileFindCloseEx() , FileNameFoundEx() , FileAttrFoundEx()

Example

```
Function void FindFile()
        String CurrPath;
        String CurrFile;
        int Handle;
        CurrPath=GetProjectPath()+"\CSV\*.csv";
        Handle=FileFindFirstEx(CurrPath);
        if (Handle!=0) then
                CurrFile=FileNameFoundEx(Handle);
                if (CurrFile!="") then
                        MessageBox("File Foud: "+CurrFile,"FileNameFoudEx");
                        while (FileFindNextEx(Handle)!=false)
                                CurrFile=FileNameFoundEx(Handle);
                                MessageBox("File Foud: "+CurrFile,"FileNameFoudEx");
                        end
                end
                FileFindCloseEx(Handle);
        end
end
```

9.6.14 FileGetAttr

Description
it returns attribute of a file

Syntax
`Int FileGetAttr(String FileName)`

Parameters
FileName name of the file

Returned value
attribute of the file

Related functions
FileSetAttr()

Example
```
FileSetAttr("C:\Temporan\TryMe.bat",FileGetAttr("C:\Temporan\TryMe.bat") |
1);
// this file must be READ ONLY!
```

9.6.15 FileSetAttr

Description
it sets attributes of a file

Syntax
```
Int FileGetAttr(String FileName,Int NewAttr)
```

Parameters
FileName name of the file
NewAttr new attributes for the file

Returned value
0 (attributes successfully changed)
nonzero (onerror)
Related functions
FileGetAttr()

Example
```
FileSetAttr("C:\Temporan\TryMe.bat",FileGetAttr("C:\Temporan\
TryMe.bat")|1);
// this file must be READ ONLY
```

9.6.16 FileGetSize

Description
it returns size of a file

Syntax
```
Int FileGetSize(String FileName)
```

Parameters
FileName name of the file

Returned value
file size in bytes

Related functions
-

Example
```
CCDim=FileGetSize("C:\Temporan\Text.txt");
```

9.6.17 FileMove

Description
it moves a file

Syntax

```
Bool FileMove(String From, String To)
```

Parameters
From name of the file you wanto to move
To new destination

Returned value
true (on success)
false (on failure)

Related functions
FileCopy()

Example
```
if (FileExist("C:\Temporan\Promem.txt")) then
      FileMove("C:\Temporan\Promem.txt","E:\Temp\Promem.txt");
end
```

9.6.18 FileNameFound

(OBSOLETE)
It's recommended to use FileNameFoundEx().

Description
It returns the file name of last file read by *FileFindFirst* or *FileFindNext*

Syntax
```
String FileNameFound()
```

Parameters
-

Returned value
name of the file

Related functions
FileFindNext(), FileFindClose(), FileFindFirst()

Example
```
Function void ExampleFindFirst()
       String CurrPath;
       String CurrFile;
       CurrPath=DirectoryGetCurrent()+"\*.*";
       if (FileFindFirst(CurrPath)==0) then
              CurrFile=FileNameFound();
              MessageBox(CurrFile,CurrPath);
              while (FileFindNext() == 0)
                     CurrFile=FileNameFound();
                     MessageBox(CurrFile,CurrPath);
              end
       end
       FileFindClose();
End
```

9.6.19 FileNameFoundEx

Description
It returns the file name of last file read by *FileFindFirstEx* or *FileFindNextEx*

Syntax
`String FileNameFoundEx(int Handle)`

Parameters
Handle this is the handle of the search in progress.

Returned value
name of the file

Related functions
FileFindFirstEx() , FileFindNextEx() , FileFindCloseEx() , FileAttrFoundEx()

Example
```
Function void FindFile()
	String CurrPath;
	String CurrFile;
	int Handle;
	CurrPath=GetProjectPath()+"\CSV\*.csv";
	Handle=FileFindFirstEx(CurrPath);
	if (Handle!=0) then
		CurrFile=FileNameFoundEx(Handle);
		if (CurrFile!="") then
			MessageBox("File Foud: "+CurrFile,"FileNameFoudEx");
			while (FileFindNextEx(Handle)!=false)
				CurrFile=FileNameFoundEx(Handle);
				MessageBox("File Foud: "+CurrFile,"FileNameFoudEx");
			end
		end
		FileFindCloseEx(Handle);
	end
end
```

9.6.20 FileOpen

Description
it opens a file with specified mode

Syntax
`int FileOpen(String FileName, String Mode)`

Parameters
FileName name of the file
Mode open mode (see modes below)

Returned value
handle of the file (on success)
0 (on failure)

Related functions
FileClose()

Example
```
FileHandle=FileOpen("C:\Temporan\Promem.txt","wt"); // open a text file for
writing
FileWriteLn(FileHandle,"Stringa di prova");
FileWriteLn(FileHandle,"questa è la seconda riga");
FileClose(FileHandle);
```

Modes:

Value	Description
r	Open file for read only
w	Open file for writing, if it doesn't exist then it will be created else it will be overwritten
a	Open file for appending, if it doesn't exist then it will be created
r+	Open file for updating (reading and writing)
w+	Make a new file ready for reading and writing, if it exists then it will be overwritten
a+	Open file for appending or creates it (if not exist); file is ready for reading and writing

If you add **t** or **b** into the *mode* string, you can specify the data type you work on, **t**ext or **b**inary data.

9.6.21 FilePos

Description
it returns the current file pointer position (file has to be opened)

Syntax
```
Int FilePos(Int Handle)
```

Parameters
Handle handle of opened file

Returned value
position of the reading/writing file pointer

Related functions
FileSeek()

Example
```
CurPos=FilePos(CurrHandle);
```

9.6.22 FileRead

Description
it reads chars form a binary file

Syntax
```
String FileRead(Int Handle, Int Length)
```

Parameters
Handle handle of opened binary file
Length number of chars tu read

Returned value
chars read from file

Related functions
FileWrite(), FileSize()

Example
```
Buffer=FileRead(FileHandle,16);  // it reads 16 chars
```

9.6.23 FileReadLn

Description
it reads a line form a text file

Syntax
String FileReadLn(Int Handle)

Parameters
```
Handle    handle of opened text file
```

Returned value
line read from file

Related functions
FileWriteLn(), FileOEF()

Example
```
CurrLine=FileReadLn(TextFileHandle);
```

9.6.24 FileRename

Description
it changes the name of a file

Syntax
```
Int FileRename(String OldName, String NewName)
```

Parameters
OldName filename of the existing file
NewName new file name for the file

Returned value
0 (on success)
-1 (on error)

Related functions
-

Example
```
if (FileExist("C:\Temporan\Promem.bak"))then
      FileRename("C:\Temporan\Promem.bak","C:\Temporan\Promem.txt");
end
```

9.6.25 FileSeek

Description

Moves the reading/writing pointer

Syntax
```
int FileSeek(Int Handle, Int Offset, Int Whence)
```

Parameters
Handle handle of the file
Offset position of the pointer
Whence pointer movement from here (see table below)

Whence Table

Value	Description
0	it sets position from start of file
1	it sets position from current pos.
2	it sets position from end of file

Returned value
0 on success (otherwise failure)

Related functions
FilePos()

Example
```
FileSeek(FileHandle, 15,0); // pos 15 from file start
```

9.6.26 FileSize

Description
it returns the size of an opened file

Syntax
```
Int FileSize(Int Handle)
```

Parameters
Handle handle of the opened file

Returned value
file size in bytes

Related functions
-

Example
```
DirSize=DirSize+FileSize(CurrFileHandle);
```

9.6.27 FileSplitPath

Description
it splits a complete file name into its components (Drive, Path,FileName and Extension)

Syntax
```
String FileSplitPath(String FileName, Int Elem)
```

Parameters

FileName	complete file name
Elem	requested component: 0 - Drive 1 - Path 2 - FileName 3 - Extension

Returned value
requested element

Related functions

-

Example
```
String FN="c:\windows\system\win.ini";
String Drive;
String Path;
String FileName;
String Ext;

Drive=FileSplitPath(FN,0);   // C
Path=FileSplitPath(FN,1);   // \WINDOWS\SYSTEM
FileName=FileSplitPath(FN,2);   // WIN
Ext=FileSplitPath(FN,3); // .ini
```

9.6.28 FileWrite

Description
it writes chars in a binary file

Syntax
```
int FileWrite(Int Handle, String Data, Int Length)
```

Parameters

Handle	handle of opened binary file
Data	chars to write
Length	number of chars to write

Returned value
number of chares that are been written

Related functions
FileRead(), FileSize()

Example
```
FileWrite(FileHandle,Buffer,16); // it writes 16 chars from Buffer to file
```

9.6.29 FileWriteLn

Description
it writes a line in a text file

Syntax
```
Int FileWriteLn(Int Handle, String Line)
```

Parameters

Handle	handle of opened text file
Line	string to write

Returned value
non negative value on success

Related functions
FileReadLn(), FileOEF()

Example
```
FileWriteLn(TextFileHandle,UserInserted);
```

9.7 Gates

9.7.1 NumGates

9.7.1.1 GetNumGateCommunicationStatus

Description
Returns the status of the specified gate

Syntax
```
Bool GetNumGateCommunicationStatus(String Name, Int Id)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)

Returned value
true (if gate is OK)
false (if gate is KO)

Related functions
GetDigGateCommunicationStatus(), GetStrGateCommunicationStatus()

Example
```
Bool ok;
ok = GetNumGateCommunicationStatus ("NUM 100",1);
```

9.7.1.2 GetNumGateGateID

Description
Returns the name ("*Gate ID*" field in *GateBuilder*) of the indicated numeric gate.

Syntax
String GetNumGateGateID(Int Index)

Parameters
Index gate index in the numeric gates list.

Returned value
GateID as string value.

Related functions
GetNumGateNID()

Example
```
string GateID;
GateID = GetNumGateGateID (10);
```

9.7.1.3 GetNumGateNID

Description
Returns the identifier ("*N ID*" field in *GateBuilder*) of the indicated numeric gate.

Syntax
Int GetNumGateNID(Int Index)

Parameters

Index gate index in the numeric gates list.

Returned value
Gate NID as integer value.

Related functions
GetNumGateGateID()

Example
```
int GateNID;
GateNID = GetNumGateNID (10);
```

9.7.1.4 GetNumGateProp

Description
Returns a numeric gate property into a string

Syntax
```
String GetNumGateProp(String Name, Int Id, int Property)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)
Property property number

PROPERTIES	***CONSTANT***
Channel number	0
Device number	1
Gate address	2
Value type	3
Decimals	4
Min value	5
Max value	6
Multiply factor	7
Addictive factor	8
Tolerance	9
Sampling rate	10
Read status	11
Write status	12

Returned value
String containing the requested property

Related functions
-

Example
```
samplingRate = GetnumeGateProp (NUM1, ID1, 10);
```

9.7.1.5 GetNumGateValue

Description
It returns the value of the specified gate.

Syntax
```
Real GetNumGateValue(String Name, Int Id)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)

Id gate identifier ("*N ID*" field in *GateBuilder*)

Returned value
value of the specified gate

Related functions
SetNumGateValue()

Example
```
int Gate12Id;
String Gate12Name;
Gate12 = GetNumGateValue(Gate12Name, Gate12Id);
```

9.7.1.6 GetNumGateValueAsString

Description
It returns the value of the specified gate as a formatted text string.

Syntax
```
String GetNumGateValueAsString(String Name, Int Id,String Format)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)
Format format of the returned text.
Using the "%g" format, the software will show the real number using the less space possible (if needed using the exponential notation). If you want to specify how the number should appear, is possible to use the notation "%x.ylf", where:
·x is a number (optional), and indicates the number of digits to show. If it is not present, all digits from the value read from the gate will be shown. If it is preceded by a 0, some 0 before the number will be shown in order to reach the specified number of digits.
·y is a number (optional), and indicates the number decimal digits to show
If y is equal to "*" then the number of decimal digits to show is the number of decimal digits defined for the given gate (this number is defined using Gate Builder).

Some examples:
"%5.2lf" will produce 123.45
"%5.0lf" will produce 123
"%07.2lf" will produce 00123.45
"%7.*lf" will produce 123.456 if the number of decimal digits defined with Gate Builder for the gate associated to the label object is equal to 3.

In the same way it is possible to specify the format for integer number ("%xd") . The meaning of the x parameter is the same described above.

Returned value
value of the specified gate as formatted text

Related functions
GetNumGateValue()

Example
```
int Gate12Id;
String Gate12Name;
Sring Value;
Value = GetNumGateValueAsString(Gate12Name, Gate12Id,"%6.*lf");
```

9.7.1.7 GetTotalNumGates

Description
Returns the number of numeric gates defined in the application.

Syntax
```
int  GetTotalNumGates()
```

Parameters
-
Returned value
The number of numeric gates defined in the application.

Related functions
GetTotalDigGates(),GetTotalEvnGates(),GetTotalCmpGates(),GetTotalStrGates()

Example
```
TotalGates=GetTotalNumGates();
```

9.7.1.8 NumGateExists

Description
Check if the specified numeric gate exists.

Syntax
```
Bool NumGateExists(String Name, Int Id)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)

Returned value
True if the gate is defined in the application
False if the gate does not defined in the application

Related functions
CmpGateExists() , DigGateExists() , EvnGateExists() , StrGateExists()

Example
```
Bool ok;
ok = NumGateExists("N",1);
```

9.7.1.9 SetNumGateInMonitor

Description
Enable/disable sampling of numeric gates defined as sampling "if in monitor"

Syntax
```
Bool SetNumGateInMonitor(String Name, Int Id, Bool Enable)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)
Enable "true" enable sampling of the gate,"false" disable sampling of the gate.

Returned value
true (if the gate is existing)
false (if not)

Related functions
SetDigGateInMonitor(),SetStrGateInMonitor()

Example
```
SetNumGateInMonitor(PrimaryName,PrimaryID,true);
```

9.7.1.10 SetNumGateValue

Description
It changes the value of the specified gate.

Syntax
```
Bool SetNumGateValue(String Name, Int Id, Real Value)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)
Value numerical value you want to give to the gate

Returned value
true (if the gate is existing and has been changed)
false (if not)

Related functions
GetNumGateValue()

Example
```
real Value = 123;
SetNumGateValue("SetPoint",1, Value);
```

9.7.2 DigGates

9.7.2.1 DigGateExists

Description
Check if the specified digital gate exists.

Syntax
```
Bool DigGateExists(String Name, Int Id)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)

Returned value
True if the gate is defined in the application
False if the gate does not defined in the application

Related functions
CmpGateExists() , EvnGateExist() , NumGateExists() , StrGateExists()

Example
```
Bool Exists;
Exists = DigGateExists("D",1);
```

9.7.2.2 GetDigGateCommunicationStatus

Description
Returns the status of the specified gate

Syntax
```
Bool GetDigGateCommunicationStatus(String Name, Int Id)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)

Returned value
true (if gate is OK)
false (if gate is KO)

Related functions
GetNumGateCommunicationStatus(), GetStrGateCommunicationStatus()

Example
```
Bool ok;
ok = GetDigGateCommunicationStatus ("DIG 100",1);
```

9.7.2.3 GetDigGateGateID

Description
Returns the name ("*Gate ID*" field in *GateBuilder*) of the indicated digital gate.

Syntax
String GetDigGateGateID(Int Index)

Parameters
Index gate index in the digital gates list.

Returned value
GateID as string value.

Related functions
GetDigGateNID()

Example
```
string GateID;
GateID = GetDigGateGateID (10);
```

9.7.2.4 GetDigGateNID

Description
Returns the identifier ("*N ID*" field in *GateBuilder*) of the indicated digital gate.

Syntax
Int GetDigGateNID(Int Index)

Parameters
Index gate index in the digital gates list.

Returned value
Gate NID as integer value.

Related functions
GetDigGateGateID()

Example
```
int GateNID;
GateNID = GetDigGateNID (10);
```

9.7.2.5 GetDigGateProp

Description
Returns a digital gate property into a string

Syntax
```
String GetDigGateProp(String Name, Int Id, int Property)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)
Property property number

PROPERTIES	***CONSTANT***
Channel number	0
Device number	1
Gate address	2
Sampling rate	10
Read status	11
Write status	12

Returned value
String containing the requested property

Related functions
-
Example
```
samplingRate = GetDigGateProp (DIG1, ID1, 10);
```

9.7.2.6 GetDigGateValue

Description
It returns the value of the specified digital gate.

Syntax
```
Int GetDigGateValue(String Name, Int Id)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)

Returned value
value (0 or 1) of the specified gate

Related functions
SetDigGateValue()

Example
```
DigiGate0001= GetDigGateValue("Gate0001",1);
```

9.7.2.7 GetTotalDigGates

Description
Returns the number of digital gates defined in the application.

Syntax
```
int  GetTotalDigGates()
```

Parameters
-

Returned value
The number of digital gates defined in the application.

Related functions
GetTotalCmpGates(),GetTotalEvnGates(),GetTotalNumGates(),GetTotalStrGates()

Example
```
TotalGates=GetTotalDigGates();
```

9.7.2.8 SetDigGateInMonitor

Description
Enable/disable sampling of digital gates defined as sampling "if in monitor"

Syntax
```
Bool SetDigGateInMonitor(String Name, Int Id, Bool Enable)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)
Enable "true" enable sampling of the gate,"false" disable sampling of the gate.

Returned value
true (if the gate is existing)
false (if not)

Related functions
SetNumGateInMonitor(),SetStrGateInMonitor()

Example
```
SetDigGateInMonitor(PrimaryName,PrimaryID,true);
```

9.7.2.9 SetDigGateValue

Description
It changes the value of the specified gate.

Syntax
```
Bool SetDigGateValue(String Name, Int Id, Int Value)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)
Value numerical value you want to give to the gate (0 or 1)

Returned value
true (if the gate is existing and has been changed)
false (if not)

Related functions
GetDigGateValue()

Example
```
SetDigGateValue("Flag",1, 1);
```

9.7.3 CmpGates

9.7.3.1 CmpGateExists

Description
Check if the specified compound gate exists.

Syntax
```
Bool CmpGateExists(String Name, Int Id)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)

Returned value
True if the gate is defined in the application
False if the gate does not defined in the application

Related functions
DigGateExists() , EvnGateExists() , NumGateExists() , StrGateExists()

Example
```
Bool Exists;
Exists = CmpGateExists("C",1);
```

9.7.3.2 GetCmpGateGateID

Description
Returns the name ("*Gate ID*" field in *GateBuilder*) of the indicated compound gate.

Syntax
String GetCmpGateGateID(Int Index)

Parameters
Index gate index in the compound gates list.

Returned value
GateID as string value.

Related functions
GetCmpGateNID()

Example
```
string GateID;
GateID = GetCmpGateGateID (10);
```

9.7.3.3 GetCmpGateNID

Description
Returns the identifier ("*N ID*" field in *GateBuilder*) of the compound numeric gate.

Syntax
Int GetCmpGateNID(Int Index)

Parameters
Index gate index in the compound gates list.

Returned value
Gate NID as integer value.

Related functions
GetCmpGateGateID()

Example
```
int GateNID;
GateNID = GetCmpGateNID (10);
```

9.7.3.4 GetCmpGateValue

Description
It returns the value of the specified compound gate.

Syntax
```
Real GetCmpGateValue(String Name, Int Id)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)

Returned value
value of the specified gate

Related functions
-

Example
```
String PortName;
GateName="GateA";
CurrValue= GetCmpGateValue(GateName,GateNo);
```

9.7.3.5 GetCmpGateValueAsString

Description
It returns the value of the specified gate as a formatted text string.

Syntax
```
String GetCmpGateValueAsString(String Name, Int Id,String Format)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)
Format format of the returned text.
Using the “%g” format, the software will show the real number using the less space possible (if needed using the exponential notation). If you want to specify how the number

should appear, is possible to use the notation “%x.ylf”, where:
·x is a number (optional), and indicates the number of digits to show. If it is not present, all digits from the value read from the gate will be shown. If it is preceded by a 0, some 0 before the number will be shown in order to reach the specified number of digits.
·y is a number (optional), and indicates the number decimal digits to show.
If y is equal to "*" then the number of decimal digits to show is the number of decimal digits defined for the given gate (this number is defined using Gate Builder).

Some examples:
“%5.2lf” will produce 123.45
“%5.0lf” will produce 123
“%07.2lf” will produce 00123.45
"%7.*lf" will produce 123.456 if the number of decimal digits defined with Gate Builder for the gate associated to the label object is equal to 3.

In the same way it is possible to specify the format for integer number (“%xd”) . The meaning of the x parameter is the same described above.

Returned value
value of the specified gate as formatted text

Related functions
GetCmpGateValue()

Example
```
int Gate12Id;
String Gate12Name;
Sring Value;
Value = GetCmpGateValueAsString(Gate12Name, Gate12Id,"%6.*lf");
```

9.7.3.6 GetTotalCmpGates

Description
Returns the number of compound gates defined in the application.

Syntax
```
int  GetTotalCmpGates()
```

Parameters
-

Returned value
The number of compound gates defined in the application.

Related functions
GetTotalDigGates(),GetTotalEvnGates(),GetTotalNumGates(),GetTotalStrGates()

Example
```
TotalGates=GetTotalCmpGates();
```

9.7.4 StrGates

9.7.4.1 GetStrGateCommunicationStatus

Description
Returns the status of the specified gate.

Syntax
```
Bool GetStrGateCommunicationStatus(String Name, Int Id)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)

Returned value
true (if gate is OK)
false (if gate is KO)

Related functions
GetNumGateCommunicationStatus(),GetDigGateCommunicationStatus()

Example
```
Bool ok;
ok = GetStrGateCommunicationStatus ("S",1);
```

9.7.4.2 GetStrGateGateID

Description
Returns the name ("*Gate ID*" field in *GateBuilder*) of the indicated string gate.

Syntax
String GetStrGateGateID(Int Index)

Parameters
Index gate index in the string gates list.

Returned value
GateID as string value.

Related functions
GetStrGateNID()

Example
```
string GateID;
GateID = GetStrGateGateID (10);
```

9.7.4.3 GetStrGateNID

Description
Returns the identifier ("*N ID*" field in *GateBuilder*) of the indicated string gate.

Syntax
Int GetStrGateNID(Int Index)

Parameters
Index gate index in the string gates list.

Returned value
Gate NID as integer value.

Related functions
GetStrGateGateID()

Example
```
int GateNID;
```

```
GateNID = GetStrGateNID (10);
```

9.7.4.4 GetStrGateValue

Description
It returns the value of the specified string gate.

Syntax
```
String GetStrGateValue(String Name, Int Id)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)

Returned value
string of the specified gate

Related functions
SetStrGateValue()

Example
```
CurrValue= GetStrGateValue("STRGate001",STRGateNo);
```

9.7.4.5 GetStrGateProp

Description
Returns a string gate property into a string

Syntax
```
String GetStrGateProp(String Name, Int Id, int Property)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)
Property property number

PROPERTIES	***CONSTANT***
Channel number	0
Device number	1
Gate address	2
Sampling rate	10
Read status	11
Write status	12

Returned value
String containing the requested property

Related functions
-

Example
```
samplingRate = GetStrGateProp (STR1, ID1, 10);
```

9.7.4.6 GetTotalStrGates

Description
Returns the number of string gates defined in the application.

Syntax
```
int  GetTotalStrGates()
```

Parameters
-

Returned value
The number of string gates defined in the application.

Related functions
GetTotalDigGates(),GetTotalEvnGates(),GetTotalNumGates(),GetTotalCmpGates()

Example
```
TotalGates=GetTotalStrGates();
```

9.7.4.7 StrGateExists

Description
Check if the specified string gate exists.

Syntax
```
Bool StrGateExists(String Name, Int Id)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)

Returned value
True if the gate is defined in the application
False if the gate does not defined in the application

Related functions
CmpGateExists() , DigGateExists() , EvnGateExists() , NumGateExists()

Example
```
Bool Exists;
Exists = StrGateExists("S",1);
```

9.7.4.8 SetStrGateInMonitor

Description
Enable / disable sampling of string gates defined as sampling "if in monitor"

Syntax
```
Bool SetStrGateInMonitor(String Name, Int Id, Bool Enable)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)
Enable "true" enable sampling of the gate,"false" disable sampling of the gate.

Returned value
true (if the gate is existing)
false (if not)

Related functions
SetNumGateInMonitor(),SetDigGateInMonitor()

Example

```
SetStrGateInMonitor(PrimaryName,PrimaryID,true);
```

9.7.4.9 SetStrGateValue

Description
It changes the value of the specified string gate.

Syntax
```
Bool SetStrGateValue(String Name, Int Id, String Value)
```

Parameters
Name gate name ("*Gate ID*" field in *GateBuilder*)
Id gate identifier ("*N ID*" field in *GateBuilder*)
Value new value to give to the gate

Returned value
true (if the gate is existing and has been changed)
false (if not)

Related functions
GetStrGateValue()

Example
```
GateModified= SetStrGateValue("Gate0001SS",1,"Alfa");
```

9.7.5 EvnGates

9.7.5.1 EvnGateExists

Description
Check if the specified event gate exists.

Syntax
```
Bool EvnGateExists(String Name, Int Id)
```

Parameters
Name gate name ("*Name*" field in *GateBuilder*)
Id gate identifier ("*ID*" field in *GateBuilder*)

Returned value
True if the gate is defined in the application
False if the gate does not defined in the application

Related functions
CmpGateExists() , DigGateExists() , NumGateExists() , StrGateExists()

Example
```
Bool Exists;
Exists = EvnGateExists("E",1);
```

9.7.5.2 GetEvnGateAckedStatus

Description:
It returns the acquisition state of the specified event gate.

Syntax:
```
Bool GetEvnGateAckedStatus(String Name, Int Id)
```

Parameters:
Name gate name ("*Name*" field in *GateBuilder*)
Id gate identifier ("*ID*" field in *GateBuilder*)

Returned value:
True = alarm has been acknowledged by operator
False = alarm has NOT been acknowledged yet

Related functions:
SetEvnGateAckedStatus()

Example:
```
if(GetEvnGateAckedStatus(EVN3,ID3)==false) then
   GetStrGateMsg(EV3,ID3);
end
```

9.7.5.3 GetEvnGateExcludedStatus

Description:
It returns the alarm excluded state of the specified event gate.

Syntax:
```
Bool GetEvnGateExcludedStatus(String Name, Int Id)
```

Parameters:
Name gate name ("*Name*" field in *GateBuilder*)
Id gate identifier ("*ID*" field in *GateBuilder*)

Returned value:
True = alarm has been excluded
False = alarm has NOT been excluded

Related functions:
SetEvnGateExcludedStatus()

Example:
```
if(GetEvnGateExcludedStatus(EVN3,ID3)==false) then
   GetStrGateMsg(EV3,ID3);
end
```

9.7.5.4 GetEvnGateGateID

Description
Returns the name ("*Name*" field in *GateBuilder*) of the indicated event gate .

Syntax
String GetEvnGateGateID(Int Index)

Parameters
Index gate index in the event gates list.

Returned value
String value containing the gate name.

Related functions
GetEvnGateNID()

Example

```
string GateID;
GateID = GetEvnGateGateID (10);
```

9.7.5.5 GetEvnGateMsg

Description:
Returns the message of the specified event gate.

Syntax:
```
String GetEvnGateMsg(String Name, Int Id)
```

Parameters:
Name gate name ("*Name*" field in *GateBuilder*)
Id gate identifier ("*ID*" field in *GateBuilder*)

Valore restituito:
message of the event gate

Funzioni inerenti:
GetEvnGateValue()

Example:
```
GateA_Msg = GetEvnGateMsg(GateA_Name,1);
```

9.7.5.6 GetEvnGateNID

Description
Returns the identifier ("*ID*" field in *GateBuilder*) of the indicated event gate

Syntax
Int GetEvnGateNID(Int Index)

Parameters
Index gate index in the event gates list.

Returned value
Integer value containing the gate indentifier.

Related functions
GetEvnGateGateID()

Example
```
int GateNID;
GateNID = GetEvnGateNID (10);
```

9.7.5.7 GetEvnGateValue

Description
It returns the value of the specified event gate.

Syntax
```
Bool GetEvnGateValue(String Name, Int Id)
```

Parameters
Name gate name ("*Name*" field in *GateBuilder*)
Id gate identifier ("*ID*" field in *GateBuilder*)

Returned value

- *true* : if the event is active
- *false*: if the event is NOT active.

Related functions
GetEvnGateMsg()

Example
```
GateModified= GetEvnGateValue("Gate0001",1);
```

9.7.5.8 GetEvnGateSignificantStatus

Description:
It returns the "*significant*" status of the specified event gate.
Event is considered "*significant*" if it is not excluded and if:
- it is active but not acked yet (in case of "retained" event)
- no more active but not acked yet (in case of "retained" event)

Syntax:
```
Bool GetEvnGateSignificantStatus(String Name, Int Id)
```

Parameters:

Name	gate name ("*Name*" field in *GateBuilder*)
Id	gate identifier ("*ID*" field in *GateBuilder*)

Returned value:
True = alarm is significant
False = alarm is NOT significant

Example:
```
if(GetEvnGateSignificantStatus(EVN3,ID3)==true) then
   GetStrGateMsg(EV3,ID3);
end
```

9.7.5.9 GetTotalEventGates

Description
Returns the number of event gates defined in the application.

Syntax
```
int  GetTotalEvnGates()
```

Parameters
-

Returned value
The number of event/alarm gates defined in the application.

Related functions
GetTotalDigGates(),GetTotalCmpGates(),GetTotalNumGates(),GetTotalStrGates()

Example
```
TotalGates=GetTotalEvnGates();
```

9.7.5.10 SetEvnGateAckedStatus

Description:
Set as "Acked" the specified event gate.

Syntax:
```
Bool SetEvnGateAckedStatus(String Name, Int Id)
```

Parameters:
Name gate name ("*Name*" field in *GateBuilder*)
Id gate identifier ("*ID*" field in *GateBuilder*)

Returned value:
True = operation completed
False = gate not found

Related functions:
GetEvnGateAckedStatus()

Example:
```
SetEvnGateAckedStatus("GateTest",1);
```

9.7.5.11 SetEvnGateExcludedStatus

Description:
Excludes / Includes the specified event gate.

Syntax:
```
Bool SetEvnGateExcludedStatus(String Name, Int Id)
```

Parameters:
Name gate name ("*Name*" field in *GateBuilder*)
Id gate identifier ("*ID*" field in *GateBuilder*)
Exclude True : it indicates that the alarm must be excluded from control
False: it indicates that the alarm must NOT be excluded from the control

Returned value:
True = operation completed
False = gate not found

Related functions:
GetEvnGateExcludedStatus()

Example:
```
SetEvnGateExcludedStatus("GateTest",1,true);
```

9.8 Generic

9.8.1 AppendUserChangesEntry

Description
Append a new record to the User Changes Historical file.

Syntax
Void AppendUserChangesEntry(String Code,String Message)

Parameters
Code message to write in the "Code" column.
Message message to write in the "Message" column.

Returned value
-
Related functions
-

Example
```
AppendUserChangesEntry("Section1","Machine stopped by user");
```

9.8.2 Beep

Description
It sends out an acoustic signal.

Syntax
```
Void Beep()
```

Parameters
-

Returned value
-

Related functions
-

Example
```
Beep();
```

9.8.3 CloseKeyboard

Description
Close the keyboard actually opened.

Syntax
Void CloseKeyboard()

Parameters
-

Returned value
None

Related functions
Keyboard()

Example
```
CloseKeyboard();.
```

9.8.4 CloseSession

Description
Already open session is closed

Syntax
```
Void CloseSession()
```

Parameters
-

Returned value
(none)

Related functions
-

Example
```
CloseSession();
```

9.8.5 CloseWindow

Description
It closes the window from which the function is called.

Syntax
```
Void CloseWindow()
```

Parameters
-

Returned value
-

Related functions
WindowIsOpen()

Example
```
CloseWindow();
```

9.8.6 EnableShutdown

Description
Enable / Disable computer automatic power off after an exit of supervision session.

Syntax
```
Void EnableShutdown(Bool Flag)
```

Parameters
Flag
true for automatic computer power off on supervision session exit.
false for no computer poer off on supervision session exit.

Returned value
-

Related functions
GetShutdownStatus()

Example
```
Function void DisablePCOff()
   EnableShutdown(false);                                    // Computer
power off disabled
end


Function void EnablePCOff()
```

```
   EnableShutdown(true);                                    // Computer
power off enabled
end
```

9.8.7 Exec

Description
Executes a program

Syntax
`Int Exec(String ProgName)`

Parameters
ProgName filename of the program

Returned value
value > 31 (on success)

Related functions
-

Example
```
if (Exec("c:\windows\command\command.com") > 31) then
      DosShellExecuted=true;
end
```

9.8.8 ExecEx

Description
Executes a program

Syntax
`Int ExecEx(String ProgName,int WindowType)`

Parameters
ProgName filename of the program
WindowType
If 1 then executes a program in a Maximized window
If 2 then executes a program in a Minimized window
If 3 then executes a program in a default size window

Returned value
value > 31 (on success)

Related functions
-

Example
```
if (ExecEx("c:\windows\command\command.com",1) > 31) then
      DosShellExecuted=true;
end
```

9.8.9 FileOpenDialog

Description
Open a standard FileOpen dialog.

Syntax

```
String FileOpenDialog(
                        string Filename ,
                        string FilenameFilter,
                        int FilterIndex,
                        string InitialDir,
                        string DefaultExt,
                        bool HideReadOnly,
                        bool PathMustExist,
                        bool FileMustExist,
                        bool NoValidate,
                        bool NoChangeDir,
                        bool AllowMultiSelect,
                        bool CreatePrompt,
                        bool NoReadOnlyReturn,
                        bool NoTestFileCreate,
                        bool OverwritePrompt,
                        bool ShareAware,
                        bool ShowHelp)
```

Parameters

string FileName	name and directory path of the last file selected.
string FileNameFilter	file masks (filters)
int FilterIndex	filter selected by default when the dialog opens.
string InitialDir	current directory when the dialog opens.
string DefaultExt	default file extension.
bool HideReadOnly	removes the Open As Read Only check box from the dialog.
bool PathMustExist	generates an error message if the user tries to select a file name with a nonexistent directory path.
bool FileMustExist	generates an error message if the user tries to select a nonexistent file.
bool NoValidate	disables checking for invalid characters in file names. Allows selection of file names with invalid characters.
bool NoChangeDir	after the user clicks OK, resets the current directory to whatever it was before the file-selection dialog opened.
bool AllowMultiSelect	allows users to select more than one file in the dialog.
bool CreatePrompt	generates a warning message if the user tries to select a nonexistent file, asking whether to create a new file with the specified name.
bool NoReadOnlyReturn	generates an error message if the user tries to select a read-only file.
bool NoTestFileCreate	disables checking for network file protection and inaccessibility of disk drives. Applies only when the user tries to save a file in a create-no-modify shared network directory.
bool OverwritePrompt	generates a warning message if the user tries to select a file name that is already in use, asking whether to overwrite the existing file.

bool ShareAware	ignores sharing errors and allows files to be selected even when sharing violations occur.
bool ShowHelp	displays a Help button in the dialog.

Returned value
A string with path + file name if the "OK" button is pressed.
A string with "" if "ESC" button is pressed.

Related functions
FileSaveDialog()

Example

```
string FileName;
FileName=FileOpenDialog("",//Filename .
                        "*.txt",//Filename filter & filter patterns.
                        1,//Filter Index
                        "C:\",//Initial dir.
                        "*",//Default extension
                        true,// HideReadOnly
                        false,// PathMustExist
                        false,// FileMustExist
                        false,// NoValidate
                        false,// NoChangeDir
                        false,// AllowMultiSelect
                        false,// CreatePrompt
                        false,// NoReadOnlyReturn
                        false,// NoTestFileCreate
                        false,// OverwritePrompt
                        false,// ShareAware
                        false);// ShowHelp

MessageBox(FileName,"File Selected");
```

9.8.10 FileSaveDialog

Description
Open a standard FileSave dialog.

Syntax
String FileSaveDialog(
string Filename ,
string FilenameFilter,
int FilterIndex,
string InitialDir,
string DefaultExt,
bool HideReadOnly,
bool PathMustExist,
bool NoValidate,
bool NoChangeDir,
bool AllowMultiSelect,
bool CreatePrompt,
bool NoReadOnlyReturn,
bool NoTestFileCreate,
bool OverwritePrompt,

bool ShareAware,
bool ShowHelp)

Parameters

string FileName	name and directory path of the last file selected.
string FileNameFilter	file masks (filters)
int FilterIndex	filter selected by default when the dialog opens.
string InitialDir	current directory when the dialog opens.
string DefaultExt	default file extension.
bool HideReadOnly	removes the Open As Read Only check box from the dialog.
bool PathMustExist	generates an error message if the user tries to select a file name with a nonexistent directory path.
bool NoValidate	disables checking for invalid characters in file names. Allows selection of file names with invalid characters.
bool NoChangeDir	after the user clicks OK, resets the current directory to whatever it was before the file-selection dialog opened.
bool AllowMultiSelect	allows users to select more than one file in the dialog.
bool CreatePrompt	generates a warning message if the user tries to select a nonexistent file, asking whether to create a new file with the specified name.
bool NoReadOnlyReturn	generates an error message if the user tries to select a read-only file.
bool NoTestFileCreate	disables checking for network file protection and inaccessibility of disk drives. Applies only when the user tries to save a file in a create-no-modify shared network directory.
bool OverwritePrompt	generates a warning message if the user tries to select a file name that is already in use, asking whether to overwrite the existing file.
bool ShareAware	ignores sharing errors and allows files to be selected even when sharing violations occur.
bool ShowHelp	displays a Help button in the dialog.

Returned value
A string with path + file name if the "OK" button is pressed.
A string with "" if "ESC" button is pressed.

Related functions
FileOpenDialog()

Example

```
string FileName;

FileName=FileSaveDialog("",//Filename .
```

```
                                "*.txt",//Filename filter & filter patterns.
                                1,//Filter Index
                                "C:\",//Initial dir.
                                "*",//Default extension
                                true,// HideReadOnly
                                false,// PathMustExist
                                false,// NoValidate
                                false,// NoChangeDir
                                false,// AllowMultiSelect
                                false,// CreatePrompt
                                false,// NoReadOnlyReturn
                                false,// NoTestFileCreate
                                false,// OverwritePrompt
                                false,// ShareAware
                                false);// ShowHelp

   MessageBox(FileName,"File Selected");
```

9.8.11 GetProjectCaption

Description
Returns the current project caption

Syntax
```
String  GetProjectCaption()
```

Parameters
-

Returned value
string with the caption of current project

Related functions
GetProjectPath()

Example
```
ProjName=GetProjectCaption();
```

9.8.12 GetProjectName

Description
Returns the current project name

Syntax
```
String  GetProjectName()
```

Parameters
-

Returned value
string with the name of current project

Related functions
GetProjectPath()

Example
```
ProjName=GetProjectName();
```

9.8.13 GetProjectPath

Description
Returns the current project path

Syntax
```
String GetProjectPath()
```

Parameters
-

Returned value
string with the path of current project

Related functions
GetProjectName()

Example
```
ProjPath=GetProjectPath();
```

9.8.14 GetShutdownStatus

Description
Return the status enabled or disabled of computer automatic power off after an exit of supervision session.

Syntax
```
Bool GetShutdownStatus()
```

Parameters
-

Returned value
True : computer automatic power off is enabled.
False : computer automativ power off is disabled.

Related functions
EnableShutdown()

Example
```
Function void Main()
   if (GetShutdownStatus()==true) then
            MessageBox(0,"Power off enabled","Status",0);
   else
            MessageBox(0,"Power off disabled","Status",0);
  end
end
```

9.8.15 HexStrToInt

Description
Converts an hex number to integer

Syntax
```
Int HexStrToInt(String HexValue)
```

Parameters

HexValue string containing hex number

Returned value
the integer conversione of the hex number

Related functions
IntToHexStr()

Example
```
int RedColor;
RedColor=HexStrToInt("FF0000");
```

9.8.16 Keyboard

Description
View the keyboard specified by the name.
If the application supports several languages, then will be automatically loaded the keyboard named (name)+"_"+(current language). (for example : "AlphanumericKeyb_English")

The keyboard must be defined by Keyboard Builder tool.
This function is utilized for touch screen applications.
Following there are some keyboards examples built with Keyboard Builder

Syntax
Void Keyboard(String Name)

Parameters
Name keyboard name to open

Returned value
None

Related functions
CloseKeyboard()

Example

```
Keyboard("NumericKeyboard");
```

9.8.17 IconMessageBox

Description
It shows on the screen a window containing text and the specified title, plus any combination of predefined icons and push buttons. It waits for confirmation from the user.

Syntax
```
Int IconMessageBox(
                    String Text, String Title, Int ButtonType, Int
                    IconType, Int DefaultButton)
```

Parameters

Text	the text in the window
Title	the title of the window
ButtonType	indicate the button contained in the message box
IconType	display an icon in the message box
DefaultButton	indicate the default button

ButtonType:
1 message box contains three push buttons: Abort, Retry, Ignore
2 message box contains one push buttons: OK
3 message box contains two push buttons: OK, Cancel
4 message box contains two push buttons: Retry, Cancel
5 message box contains two push buttons: Yes, No
6 message box contains three push buttons: Yes, No, Cancel

IconType
1 exclamation-point icon appears in the message box (iconexclamation)
2 exclamation-point icon appears in the message box (iconewarning)
3 icon consisting of a lowercase letter *i* in a circle appears in the message box (iconinformation)
4 icon consisting of a lowercase letter i in a circle appears in the message box (iconasterisk)
5 question-mark icon appears in the message box (iconquestion)
6 stop-icon appears in the message box (iconstop)
7 stop-icon appears in the message box (iconerror)
8 stop-icon appears in the message box (iconhand)

DefaultButton
1 frist is the default button
2 second is the default button
3 third button is the default button
4 fourth is the default button

Returned value
0 if there is not enough memory to create the message box
1 if Abort button was selected
2 if Cancel button was selected
3 if Ignore button was selected
4 if No button was selected
5 if OK button was selected
6 if Retry button was selected
7 if Yes button was selected

Related functions
QuestionBox(), InputDialog()

Example
```
IconMessageBox("You can now switch off","Shutdown sequence:", 2, 3,1);
```

9.8.18 InputDialog

Description
It shows on the screen a dialog box containing a text, a specified title, a string field that can be changed, and some buttons to confirm or undo; a string field is a string that the user can change at will using the keyboard.

Syntax
```
String InputDialog(String Text, String Title, String InitText)
```

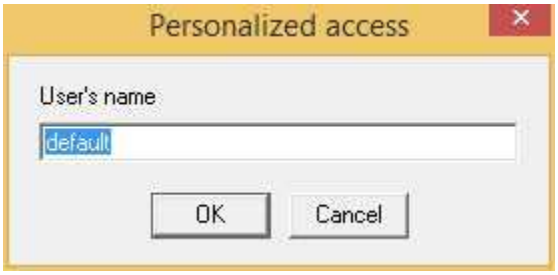
Parameters

Text	the text in the window
Title	the title of the window
InitText	the text that it is first shown in the string field

Returned value
the string field if the user confirms the change, a null string if he doesn't

Related functions
MessageBox(), QuestionBox()

Example
```
User=InputDialog("User's name", "Personalized access", "default");
```



9.8.19 IntToHexStr

Description
Converts an integer value into hex format

Syntax
```
String IntToHexStr(Int Value, Int Digits)
```

Parameters

Value	integer to be converted
Digits	minimum string size, if number converted is shorter then add '0' to the left until string length is *Digits*

Returned value
String containing the hexadecimal representation of the integer value

Related functions
HexStrToInt()

Example
```
Pattern = IntToHexStr(12800,4);
```

9.8.20 Message Beep

Description:
It sends out an specified acoustic signal.

Syntax:
```
Void MessageBeep(int type)
```

Parameters:
type numerical value that specify kind of acoustic signal expected

Value	*Acoustic message*
1	IconAsterisk
2	IconExclamation
3	IconHand
4	IconQuestion
5	OK

(if type parameter take none of this value then it play a single tone)

Returned value:
(none)

Related functions:
Beep()

Example:
```
MessageBeep(5);
```

9.8.21 MessageBox

Description
It shows on the screen a window containing text and the specified title. It waits for confirmation from the user.

Syntax
```
Void MessageBox(String Text, String Title)
```

Parameters
Text the text in the window
Title the title of the window

Returned value
-

Related functions
QuestionBox(), InputDialog()

Example
```
MessageBox("You can now switch off", "Shutdown sequence:");
```

9.8.22 Play

Description
Plays an audio file with WAV extension
It is an obolete function. Refer to PlaySound() function.

Syntax
```
Void  Play(String AudioFileName)
```

Parameters
AudioFileName filename of the audio file

Returned value
-

Related functions
PlaySound()

Example
```
Play("Z:\Audio\WAV\Alarm.wav");
```

9.8.23 PlaySound

Description
Plays an audio file with WAV or MP3 extension

Syntax
```
Void  PlaySound(String AudioFileName,bool WaitMode,int RepeatCounter)
```

Parameters

AudioFileName	filename of the audio file
WaitMode	if true the function waits untill the sound playing is terminated.
RepeatCounter	number of time that the sound will be repeated. If it is "0" then the sound will be repeated continuously and can be terminated with the StopSound() function.

Returned value
-

Related functions
StopSound()

Example
```
Play("Z:\Audio\WAV\Alarm.wav",false,0);
```

9.8.24 PrintString

Description:
Prints string on the specified device

Syntax:
```
Bool PrintString(String Msg, String Dev, Bool AddLF)
```

Parameters:

Msg	messagge to print
Dev	device to print on(LPT1, LPT2, etc)
AddLF	add LineFeed

Returned value:
true (on success)
false (on failure)

Related functions:
-

Example:
```
PrintString("INIT Failed!","LPT1",True);
```

9.8.25 QuestionBox

Description
It shows on the screen a window containing text and the specified title. It waits for the user to choose Yes or No.

Syntax
```
Bool QuestionBox(String Text, String Title)
```

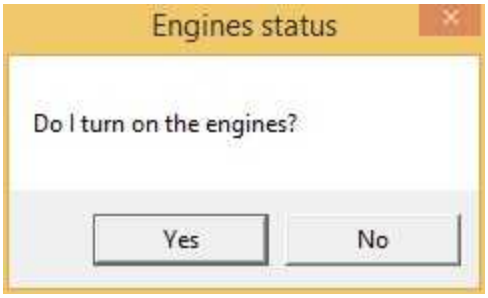
Parameters

Text	the text in the window
Title	the title of the window

Returned value
true if the user chooses Yes
false if the user chooses No

Related functions
MessageBox(), InputDialog()

Example
```
Bool Engines = QuestionBox("Do I turn on the engines?","Engines status");
```



9.8.26 RealToInt

Description
Converts a real value into an integer (discarding decimals)

Syntax
```
int RealToInt(Real Value)
```

Parameters
Value real value

Returned value
integer that is real value truncated

Related functions
-

Example
real TR = 123.45;
int Temp = RealToInt(TR);

9.8.27 ShellExec

Description
The ShellExec function performs an action on a file. The file can be an executable file or a document.

Syntax
```
Int ShellExec(String FileName,String Verb,String WorkDir,int
              ShowMode,String Class,String Args)
```

Parameters

FileName — specify the name of the file to open or print. The function can open an executable file or a document file. The function can print a document file. If the path is not included with a name, the current directory is assumed.

Verb — specifies an action for the application to perform. This member defaults to "Open" if no verb is specified.

WorkDir — specifies the name of the working directory. If this member is not specified, the current directory is used as the working directory.

ShowMode — can be one of the values described below. If FileName specifies an executable file, ShowMode specifies how the application is to be shown when it is opened. If FileName specifies a document file, ShowMode should be zero.
0 = for document file.
1 = SW_SHOW.
2 = SW_SHOWMAXIMIZED.
3 = SW_SHOWMINIMIZED.
4 = SW_HIDE.
5 = SW_MINIMIZE.
6 = SW_RESTORE.
7 = SW_SHOWMINNOACTIVE.
8 = SW_SHOWNA.
9 = SW_SHOWNOACTIVATE.
10=SW_SHOWNORMAL.

Class — specifying the name of a file class or a globally unique identifier (GUID).

Args application parameters. The parameters must be separated by spaces.

Returned value
value > 31 (on success)

Related functions
-

Example
```
ShellExec("MyDoc.pdf","open","C:\",0,".pdf","");
```

9.8.28 Sleep

Description
It waits for a specified period of time expressed in milliseconds.

Syntax
```
Void Sleep(int MilliSeconds)
```

Parameters
Milliseconds duration of the pause in milliseconds

Returned value
-

Related functions
-

Example
```
Sleep(1000);
```

9.8.29 StopFunction

Description:
Stops a running function

Syntax:
```
Int StopFunction (String FunctionName)
```

Parameters:
FunctionName name of the function

Returned value:
0 on success
-1 unknown function
-2 function is not running

Related functions:
-

Example:
```
int res;
res=StopFunction("Reader");
...
...
Function void Reader()
```

```
...
end
```

9.8.30 StopSound

Description
Stops plaing a sound started with PlaySound function

Syntax
```
Void  StopSound()
```

Parameters
-
Returned value
-

Related functions
PlaySound()

Example
```
StopSound();
```

9.8.31 WindowIsOpen

Description
It returns the status of the window from which the function is called.

Syntax
```
Bool WindowIsOpen()
```

Parameters
-

Returned value
`true` if the window (e.g. a template), from which the function has been called up, is still open.
`false` if not.

Related functions
CloseWindow()

Example
```
StatoFinestraMadre=WindowIsOpen();
```

9.9 Internet

9.9.1 SendMail

Description
Sends an e-mail with an optional attachment files.
Works on a internet connection already active.

Syntax
```
string SendMail(
                  int Timeout,
                  string HostSMTPServer,
                  string UserName,
                  string Password,
                  string AddressFrom,
```

```
                    string AddressTo,
                    string AddressCc,
                    string Subject,
                    string Body,
                    bool  AttachmentPresent,
                    string AttachmentPath)
```

Parameters

Timeout	timeout for complete the operation (ms).
HostSMTPServer	Host SMTP server - can be specified as following: ***HostSMTPServer*** *- not encrypted* connection on port 25 ***HostSMTPServer:Port*** *- not* encrypted connection on specified port ***tls://HostSMTPServer*** *- TLS* encrypted connection on port 587 (default for TLS) ***tls://HostSMTPServer:Port*** *- TLS* encrypted connection on specified port ***ssl://HostSMTPServer*** *- SSL* encrypted connection on port 465 (default for SSL) ***ssl://HostSMTPServer:Port*** *- SSL* encrypted connection on specified port
UserName	user account for the SMTP server.
UserPassword	password used for SMTP account authentication.
AddressFrom	source mail address (some SMTP servers block e-mail from non-existent address).
AddressTo	recipient mail address (you can enter multiple addresses separated by commas).
AddressCc	recipient carbon copy mail address (you can enter multiple addresses separated by commas).
Subject	mail subject.
Body	mail body.
AttachmentPresent	if "***true***" then send with attachment.
AttachmentPath	full path file name of the attachment.

If you use a GMAIL account is necessary ENABLE access to App less secure. This can be done on the account by using the feedback page https://www.google.com/settings/security/lesssecureapps
More information at https://support.google.com/accounts/answer/6010255?hl=en

Returned value

If no errors then an empty string is returned else the error message description is returned.

Example

```
Function void SendMailToGmail()
//*************************************************************************
****
// Send Mail demo procedure
//*************************************************************************
****
      string Error = SendMail(10000,
                                          "tls://smtp.gmail.com",
                                          "my.address@gmail.com",
                                          "MyPassword",
                                          "my.address@gmail.com",
                                          "recipient_1@company.com,
recipient_2@company.com",
                                          "recipient_3@company.com",
                                          "Test e-mail",
                                          "This is a test email sent by
```

```
Winlog",
                                                false, "");

      if (Error == "") then
            MessageBox("Operation completed", "Send Mail");
      else
            MessageBox(Error, "Send Mail");
      end
end
```

9.9.2 FTP

9.9.2.1 FTPConnect

Description
Open an FTP connection to a remote FTP server.

Syntax
```
int FTPConnect(
                                int FTPPort,
                                string FTPServerHost,
                                string UserName,
                                string Password,
                                bool   Passive,
                                int Timeout)
```

Parameters

FTPPort	FTP server TCP port.
FTPServerHost	FTP server IP address or Host name.
FTPServerHost	FTP server address can be specified in the following ways: **FTPServerHost** - Connection with automatic encryption, TLS connection is attempted, otherwise no encryption is used. **ftp://FTPServerHost** - Unencrypted connection. **ftps://FTPServerHost** - Connection with TLS encryption.
UserName	Username for FTP server access.
UserPassword	Password used for FTP account authentication.
Passive	FTP data connection method: **true** = "Passive" mode, **false** = "Active" mode.
Timeout	Timeout for complete the operation (ms).

Returned value

0 - 9	handle of the connection.
-1	value means that there are no connections available (the maximum number of simultaneous connections are 10).
-2	generic error

Related functions
FTPDisconnect(), FTPGet(), FTPPut(), FTPDelete(), FTPMakeDir(), FTPRemoveDir()

Example
```
Function void Connect()
//*************************************************************************
****
// FTP Connection
//*************************************************************************
****
#modal
```

```
    int Handle;
    Handle = FTPConnect(21, "ftp.test.com", "myusername", "mypassword",
false, 10000);
    if (Handle == -1) then
        MessageBox("No available connections", "Error");
    end

    if (Handle == -2) then
        MessageBox("Generic error", "Error");
    end

    SetNumGateValue("Connection", 1, Handle);
end
```

9.9.2.2 FTPDelete

Description
Delete a file from a remote FTP server.

Syntax
```
Bool FTPDelete(
             int Handle,
             string FileName)
```

Parameters

Handle	Connection handle previously opened by the function FTPConnect ().
FileName	Name of the file to be deleted on the FTP server (full path).

Returned value
True : if operation is completed successfully.
False : if there is an error.

Related functions
FTPConnect(), FTPDisconnect(), FTPGet(), FTPPut(), FTPMakeDir(), FTPRemoveDir()

Example
```
Function void Delete()
//*************************************************************************
****
// FTP Delete
//*************************************************************************
****
#modal
    bool Ret;
    int Handle;
    Handle = GetNumGateValue("Connection", 1);
    Ret = FTPDelete(Handle, "myFtpRemoteFolder/myFile.txt");

    if (Ret == false) then
        MessageBox("Error on FTP DELETE", "Error");
    else
        MessageBox("DELETE operation completed successfully", "Info");
    end
end
```

9.9.2.3 FTPDisconnect

Description
Close an FTP connection to a remote FTP server.

Syntax
```
Bool FTPDisconnect(int Handle)
```

Parameters
Handle Connection handle previously opened by the function FTPConnect ()

Returned value
True : if operation is completed successfully.
False : if there is an error.

Related functions
FTPConnect(), FTPGet(), FTPPut(), FTPDelete(), FTPMakeDir(), FTPRemoveDir()

Example
```
Function void Disconnect()
//*************************************************************************
****
// FTP Disconnection
//*************************************************************************
****
#modal
  bool Ret;
    int Handle;
    Handle = GetNumGateValue("Connection", 1);
    Ret = FTPDisconnect(Handle);

    if (Ret == false) then
        MessageBox("Disconnection error!", "Error");
    else
        MessageBox("Disconnection Ok!", "Info");
    end,"");
end
```

9.9.2.4 FTPGet

Description
Retrieves a file using FTP protocol from a remote FTP server.

Syntax
```
Bool FTPGet(
             int Handle,
             string SourceFileName,
             string DestinationFileName,
             int TransferType)
```

Parameters

Handle	Connection handle previously opened by the function FTPConnect ()
SourceFileName	Name of the source file on the FTP server (full path).
DestinationFileName	Destination file name that will be created on the local PC (with full path).
TransferType	If 0 then file transfer in Binary mode else if 1 then file transfer in ASCII mode.

Returned value
True : if operation is completed successfully.
False : if there is an error.

Related functions
FTPConnect(), FTPDisconnect(), FTPPut(), FTPDelete(), FTPMakeDir(), FTPRemoveDir()

Example
```
Function void Get()
//*************************************************************************
****
// FTP Get
//*************************************************************************
****
#modal
  bool Ret;
    int Handle;
    Handle = GetNumGateValue("Connection", 1);
    Ret = FTPGet(Handle, "myFtpRemoteFolder/myFile.txt",
                         GetProjectPath() + "\myFile.txt",
                         0);

    if (Ret == false) then
        MessageBox("Error on FTP GET", "Error");
    else
        MessageBox("GET operation completed successfully", "Info");
    end
end
```

9.9.2.5 FTPMakeDir

Description
Create a directory on a remote FTP server.

Syntax
```
Bool FTPMakeDir(
                  int Handle,
                  string DirectoryName)
```

Parameters

Handle	Connection handle previously opened by the function FTPConnect ()
DirectoryName	Name of the directory to be created on the FTP server (full path).

Returned value
True : if operation is completed successfully.
False : if there is an error.

Related functions
FTPConnect(), FTPDisconnect(), FTPGet(), FTPPut(), FTPDelete(), FTPRemoveDir()

Example
```
Function void MakeDir()
//*************************************************************************
****
```

```
// FTP MakeDir
//*************************************************************************
****
#modal
    bool Ret;
    int Handle;
    Handle = GetNumGateValue("Connection", 1);
    Ret = FTPMakeDir(Handle, "myFtpRemoteFolder/myDirectory/");

    if (Ret == false) then
        MessageBox("Error on FTP MAKE DIR", "Error");
    else
        MessageBox("MAKE DIR operation completed successfully", "Info");
    end
end
```

9.9.2.6 FTPPut

Description
Uploads a file using FTP protocol to a remote FTP server.

Syntax
```
Bool FTPPut(
             int Handle,
             string SourceFileName,
             string DestinationFileName,
             int TransferType)
```

Parameters

Handle	Connection handle previously opened by the function FTPConnect ()
SourceFileName	Name of the source file on the local PC (with full path).
DestinationFileName	Destination file name that will be created on the FTP server (with full path).
TransferType	If 0 then file transfer in Binary mode else if 1 then file transfer in ASCII mode.

Returned value
True : if operation is completed successfully.
False : if there is an error.

Related functions
FTPConnect(), FTPDisconnect(), FTPGet(), FTPDelete(), FTPMakeDir(), FTPRemoveDir()

Example
```
Function void Put()
//*************************************************************************
****
// FTP Put
//*************************************************************************
****
#modal
    bool Ret;
    int Handle;
    Handle = GetNumGateValue("Connection", 1);
    Ret = FTPPut(Handle, GetProjectPath() + "\myFile.txt",
                         "myFtpRemoteFolder/myFile.txt",
                         0);

    if (Ret == false) then
```

```
        MessageBox("Error on FTP PUT", "Error");
    else
        MessageBox("PUT operation completed successfully", "Info");
    end
end
```

9.9.2.7 FTPRemoveDir

Description
Delete a directory from a remote FTP server.

Syntax
```
Bool FTPRemoveDir(
                  int Handle,
                  string DirectoryName,
                  bool DeleteIfNotEmpty)
```

Parameters

Handle	Connection handle previously opened by the function FTPConnect ()
DirectoryName	Name of the directory to delete on the FTP server (full path).
DeleteIfNotEmpty	If **true** removes the directory even if it is not empty, if **false** removes the directory only if it is empty.

Returned value
True : if operation is completed successfully.
False : if there is an error.

Related functions
FTPConnect(), FTPDisconnect(), FTPGet(), FTPPut(), FTPDelete(), FTPMakeDir()

Example
```
Function void RemoveDir()
//*************************************************************************
****
// FTP RemoveDir
//*************************************************************************
****
#modal
    bool Ret;
    int Handle;
    Handle = GetNumGateValue("Connection", 1);
    Ret = FTPRemoveDir(Handle, "myFtpRemoteFolder/myDirectory/", true);
    if (Ret == false) then
        MessageBox("Error on FTP REMOVE DIR", "Error");
    else
        MessageBox("REMOVE DIR operation completed successfully", "Info");
    end
end
```

9.10 Math

9.10.1 Abs

Description
it returns the absolute value of a real number

Syntax
```
Real Abs(Real Value)
```

Parameters
Value number to process

Returned value
absolute value of the number

Related functions
-

Example
```
PosValue=Abs(NegValue);
```

9.10.2 ArcCos

Description
it returns inverse cosinus of a real value (value must be between -1 and 1)

Syntax
```
Real ArcCos(Real Value)
```

Parameters
Value value to process

Returned value
inverse cosine of the value

Related functions
Cos(), ArcSin()

Example
```
newval=ArcCos(value);
```

9.10.3 ArcSin

Description
it returns inverse sinus of a real value (value must be between -1 and 1)

Syntax
```
Real ArcSin(Real Value)
```

Parameters
Value value to process

Returned value
inverse sinus of the value

Related functions
ArcCos(), Sin()

Example
```
newval=ArcSin(value);
```

9.10.4 ArcTan

Description
it returns inverse tangent of a real value

Syntax
```
Real ArcTan(Real Value)
```

Parameters
Value value to process

Returned value
inverse tangent of the value

Related functions
Tan()

Example
```
Theta=ArcTan(Y/X);
```

9.10.5 Cos

Description
it returns cosinus of a real value (angle in radians)

Syntax
```
Real Cos(Real Value)
```

Parameters
Value value to process

Returned value
cosinus of the value

Related functions
Sin(), ArcCos()

Example
```
SinTheta=Sin(Theta);
```

9.10.6 Exp

Description
it returns exponential value of a real number (e^x)

Syntax
```
Real Exp(Real Value)
```

Parameters
Value value to process

Returned value
exponential value

Related functions

Log()

Example
```
Delta=Exp(2.758);
```

9.10.7 Log

Description
it returns logarithm of a real value

Syntax
```
Real Log(Real Value)
```

Parameters
Value value to process

Returned value
logarithm of the value

Related functions
Exp()

Example
```
SimilZero=Log(1.00001);
```

9.10.8 Mod

Description
it returns the division modulus of two integers

Syntax
```
Int Mod(int A, int B)
```

Parameters
A dividend
B divisor

Returned value
modulus of division

Related functions
-

Example
```
Resto=Mod(44,6);
```

9.10.9 Pow

Description
raising to power (x^n)

Syntax
```
Real Pow(Real Base, Real Exponent)
```

Parameters
Base base to be raised

Exponent exponent of the number

Returned value
number raised to power specified

Related functions
Sqrt()

Example
```
PowerOfTwo=Pow(2,8);
```

9.10.10 Rand

Description
it returns a random value generated from specified range

Syntax
```
Int Rand(Int Range)
```

Parameters
Range max value generated

Returned value
random value between 0 and (Range-1)

Related functions
-

Example
```
LottoNo=Rand(90)+1;
```

9.10.11 Round

Description
it rounds up a value to a specified decimal

Syntax
```
Real Round(Real Value, Int Decimals)
```

Parameters
Value value to be rounded
Decimals decimal for operation

Returned value
rounded value

Related functions
-

Example
```
LastValue=Round(Sampled,3);
```

9.10.12 Sin

Description
it returns sinus of a real value (angle in radians)

Syntax
```
Real Sin(Real Value)
```

Parameters
Value value to process

Returned value
sinus of the value

Related functions
Cos(), ArcSin()

Example
```
SinTheta=Sin(Theta);
```

9.10.13 Sqrt

Description
it returns square root of a real value

Syntax
```
Real Sqrt(Real Value)
```

Parameters
Value value to process

Returned value
square root of the value

Related functions
Pow()

Example
```
Val=Sqrt(A*B);
```

9.10.14 Tan

Description
it returns tangent of a real value (angles are in radians)

Syntax
```
Real Tan(Real Value)
```

Parameters
Value value to process

Returned value
tangent of the value

Related functions
ArcTan()

Example
```
TanAlfa=Tan(Beta-Gamma);
```

9.11 Modem

9.11.1 ModemAvailable

(OBSOLETE)

It's recommended to use SMSOpenChannel().

Description
It checks if a GSM Modem is available on the serial port

Syntax
```
int ModemAvailable(int SerialPortNumber)
```

Parameters
SerialPortNumber = it is the number of the serial port
1 - means serial port COM1
2 - means serial port COM2
x - generic serial port COMx

Returned value
result:
0 GSM modem replies and it is ready
1 (internal) it is not possible to create listening thread
2 (internal) it is not possible to allocate memory for buffers
-1 serial port can not be opened
-2 serial port is already opened
-3 modem replies ERROR (may be this is not a GSM modem)
-4 modem's reply is not a standard reply (it is not OK or ERROR)
-5 modem does not reply

Related functions
ModemSetPIN() , ModemSetServicesCenter() , ModemSetTextMode() , ModemSendSMS()

Example
```
if (ModemAvailable(2) !=0) then
      MessageBox("Modem on the COM2 is ready","MODEM");
end
```

9.11.2 ModemDial

Description
It calls a remote device through telephone line (you need a modem installed)

Syntax
```
Int ModemDial(int Channel, String Number, int TimeOut)
```

Parameters

Channel	the number of the communication channel (this channel must have a remote protocol selected)
Number	the telephone number to dial
TimeOut	number of seconds to wait for connecting

Returned value
result:
0 Connected

-1 Init Failed (initialization failed)
-2 Dial failed (it cannot dial a number)
-3 Busy (the line is busy)
-4 TimeOut
-5 Modem already in use
-6 Channel is not remote (the specified channel has not a remote protocol)
-7 Ghost Channel (the specified channel has not a protocol selected)

Related functions
ModemHangup()

Example
```
int res;
res=ModemDial(1,"800600600",30);
```

9.11.3 ModemHangup

Description
It closes the current remote connection

Syntax
```
void ModemHangup()
```

Parameters
-

Returned value
-

Related functions
ModemDial()

Example
```
ModemHangup();
```

9.11.4 ModemSendSMS

(OBSOLETE)
It's recommended to use SMSSend().

Description
Send a short text message to a SMS compliant receiver, a mobile phone for example

Syntax
```
int ModemSendSMS(int SerialPortNumber, string Number, string Message)
```

Parameters

SerialPortNumber	it is the number of the serial port 1 - means serial port COM1 2 - means serial port COM2 x - generic serial port COMx
Number	Number of the receiver device (it can be sent to all device that can accept SMS)
Message	text message to send

Returned value
result:
0 GSM modem replies and it is ready
1 (internal) it is not possible to create listening thread
2 (internal) it is not possible to allocate memory for buffers
-1 serial port can not be opened
-2 serial port is already opened
-3 modem replies ERROR (may be this is not a GSM modem)
-4 modem's reply is not a standard reply (it is not OK or ERROR)
-5 modem does not reply
-6 modem does not recognize command
-11 inserted number is too long (it must be 40 chars wide at most)
-12 inserted message is too long (it must be 160 chars wide at most)

Related functions
ModemAvailable() , ModemSetPIN() , ModemSetServicesCenter() , ModemSetTextMode()

Example
```
string message;
message = "ALARM: high temp" + GetNumGateValue("temp",11)+ " °C";
ModemSendSMS(1, "+393491212121", message);
```

9.11.5 ModemSetPIN

(OBSOLETE)
It's recommended to use SMSOpenChannel().

Description
Set the PIN to use the SIM card inserted in the GSM modem

Syntax
```
int ModemSetPIN(int SerialPortNumber, string PIN)
```

Parameters

SerialPortNumber	it is the number of the serial port 1 - means serial port COM1 2 - means serial port COM2 x - generic serial port COMx
PIN	PIN number used to access the SIM

Returned value
result:
0 GSM modem replies and it is ready
1 (internal) it is not possible to create listening thread
2 (internal) it is not possible to allocate memory for buffers
-1 serial port can not be opened
-2 serial port is already opened
-3 PIN Error (warning: there are only few tries)
-4 modem's reply is not a standard reply (it is not OK or ERROR)
-5 modem does not reply

Related functions
ModemAvailable() , ModemSetServicesCenter() , ModemSetTextMode() , ModemSendSMS()

Example

```
ModemSetPIN(2,"1234"); // Set 1234 PIN to use the SIM card
```

Note: It has to wait 30 seconds before using again the GSM Modem, otherwise authentication process can fail

9.11.6 ModemSetServicesCenter

(OBSOLETE)

It's recommended to use SMSOpenChannel().

Description
It is necessary to set the center of services number in order to send SMS

Syntax
```
int ModemSetServicesCenter(int SerialPortNumber, string
ServicesCenterNumber)
```

Parameters

SerialPortNumber	it is the number of the serial port 1 - means serial port COM1 2 - means serial port COM2 x - generic serial port COMx
ServicesCenterNumber	number of the services center (see note below)

Returned value
result:

0	GSM modem replies and it is ready
1	(internal) it is not possible to create listening thread
2	(internal) it is not possible to allocate memory for buffers
-1	serial port can not be opened
-2	serial port is already opened
-3	modem replies ERROR (may be this is not a GSM modem)
-4	modem's reply is not a standard reply (it is not OK or ERROR)
-5	modem does not reply
-11	Number of the centre too long (max 40 chars)

Related functions
ModemAvailable() , ModemSetPIN() , ModemSetTextMode() , ModemSendSMS()

Example
```
ModemSetServicesNumber(2,"+393492000300");
```

Note:
It is possible to read the services center number on a mobile phone searching through menus. The same number can be requested to own telephonic service provider or it can be found in the manual bought with the SIM card.

9.11.7 ModemSetTextMode

(OBSOLETE)

It's recommended to use SMSOpenChannel().

Description
Sometimes it is necessary to set the format of the SMS; this function set a text format for short messages

Syntax

```
int ModemSetTextMode(int SerialPortNumber)
```

Parameters
SerialPortNumber it is the number of the serial port
1 - means serial port COM1
2 - means serial port COM2
x - generic serial port COMx

Returned value
result:
0 GSM modem replies and it is ready
1 (internal) it is not possible to create listening thread
2 (internal) it is not possible to allocate memory for buffers
-1 serial port can not be opened
-2 serial port is already opened
-3 modem replies ERROR (may be this is not a GSM modem)
-4 modem's reply is not a standard reply (it is not OK or ERROR)
-5 modem does not reply

Related functions
ModemAvailable() , ModemSetPIN() , ModemSetServicesCenter() , ModemSendSMS()

Example
```
ModemSetPIN(2,"4488");
ModemSetServicesNumber(2,"+393492000300");
ModemSetTextMode(2);
```

9.12 Multilanguage

9.12.1 GetCurrentLanguage

Description
Returns the application current language

Syntax
```
String  GetCurrentLanguage()
```

Parameters
-

Returned value
string with the name of language currently selected in the application

Related functions
SetCurrentLanguage(),GetNextLanguage()

Example
```
string ProjLanguage;
ProjLanguage=GetCurrentLanguage();
```

9.12.2 GetNextLanguage

Description
Return the successive language to the one received in input

Syntax
```
string  GetNextLanguage(string Language)
```

Parameters
Language it is the language name from which will be found the successive in the language list defined for the application.
if an empty string is specified ("") then will be returned the first language of the list.

Returned value
The name of the next language found or an empty string if the end of the language list have been reached.

Related functions
GetCurrentLanguage(),SetCurrentLanguage()

Example
```
string Language="";
Do
      Language=GetNextLanguage(Language);
      MessageBox(Language,"Language found");
while(Language!="")
```

9.12.3 SetCurrentLanguage

Description
Set the application current language

Syntax
```
bool  SetCurrentLanguage(string Language,bool Mode)
```

Parameters
Language it is the language name to set
Mode if "true" then the set will be permanently.
if "false" then the set will be temporary (on application restart the set will be reset).

Returned value
"true" if function has been executed correctly,
"false" on error.

Related functions
GetCurrentLanguage(),GetNextLanguage()

Example
```
SetCurrentLanguage("English",true);
```

9.13 Password

9.13.1 AddUser

Description
Allow to define a new user with password and belonging groups

Syntax
```
Int AddUser(string UserName, string Password,int Groups,bool Overwrite)
```

Parameters
UserName user to create

Passoword	password associated to the user
Groups	groups associated to the user (refer to GetUserGroups() for more details)
Overwrite	true if the user will be overwritten if already present.

Returned value

0	User added successfully
1	User length is 0
2	User length exceed maximum length
3	Password length is 0
4	Password length exceed maximum length
5	No groups associated
6	Current user is not allowed to create new user.
7	User already defined.
8	Can't read/write password file

Related functions
RemoveUser()

Example

```
AddUser(GetStrGateValue("User",0),GetStrGateValue("Password",0),GetNumGateV
alue("Groups",0),true);
```

9.13.2 GetUserName

Description
It returns the current user name (the one used to be identified through the PasswordTemplate)

Syntax
```
String GetUserName()
```

Parameters
-

Returned value
Current user name

Related functions
GetUserGroups(),Logout()

Example
```
if (GetUserName() != "MASTER") then
      MessageBox("It is necessary to alert the MASTER","ALARMS");
end
```

9.13.3 GetUserGroups

Description
It returns the groups associated with the current user name

Syntax
```
Integer GetUserGroups()
```

Parameters
-

Returned value
An integer value that represents in binary format the groups associated with the current UserName

For example: if the current User is associated with groups 1,3 and 7 the function GetUserGroups() will return the following value : 69.

```
[2^00  * 1] +   (Group 1)       1+
[2^01  * 0] +   (Group 2)       0+
[2^02  * 1] +   (Group 3)       4+
[2^03  * 0] +   (Group 4)       0+
[2^04  * 0] +   (Group 5)       0+
[2^05  * 0] +   (Group 6)       0+
[2^06  * 1] +   (Group 7)       64+
[2^07  * 0] +   (Group 8)       0+
[2^08  * 0] +   (Group 9)       0+
[2^09  * 0] +   (Group 10)      0+
[2^10  * 0] +   (Group 11)      0+
[2^11  * 0] +   (Group 12)      0+
[2^12  * 0] +   (Group 13)      0+
[2^13  * 0] +   (Group 14)      0+
[2^14  * 0] +   (Group 15)      0
```

Related functions
GetUserName(),Logout()

Example
```
Groups=GetUserGroups();
```

9.13.4 Logout

Description
Current user is logout from supervisor (UserName is set up None)

Syntax
```
Void Logout()
```

Parameters
-

Returned value
(none)

Related functions
```
Login()
```

Example
```
Logout();
```

9.13.5 Login

Description
Connect a new user to the supervisor.

Syntax
```
Void Login(string User,string Password)
```

Parameters

User	user to login
Password	password associated to the user

Returned value
(none)

Related functions
`Logout()`

Example
```
Login();
```

9.13.6 RemoveUser

Description
Remove the specified user from the user list.

Syntax
```
Int RemoveUser(string UserName)
```

Parameters
UserName user to remove

Returned value
0 User removed successfully
1 User length is 0
2 User length exceed maximum length
6 Current user is not allowed to create new user.
8 Password file not found.
9 User not found

Related functions
AddUser()

Example
```
RemoveUser(GetStrGateValue("User",0));
```

9.13.7 UserFindFirst

Description
It starts user file reading process

Syntax
```
int UserFindFirst()
```

Parameters
-

Returned value
The handle for the search in progress. It must be used in the following instructions UserFindNext() and UserFindClose().
To know if at last one user have been found check that UserNameFound() returns a string different from "".

Related functions
UserFindNext() , UserFindClose() , UserNameFound() , UserGroupsFound()

Example

```
Function void FindUsers()
    int P=UserFindFirst();
    bool Found;
    if UserNameFound(P)!="" then Found=true; else Found=false;end
    While(Found==true)
        MessageBox(UserNameFound(P),UserGroupsFound(P));
        Found=UserFindNext(P);
    end
    UserFindClose(P);
end
```

9.13.8 UserFindNext

Description
It goes on with the user reading file process

Syntax
`bool UserFindNext(int Handle)`

Parameters
Handle this is the handle of the search in progress.

Returned value
true (another user found)
false (no more users)

Related functions
UserFindFirst() , UserFindClose() , UserNameFound() , UserGroupsFound()

Example

```
Function void FindUsers()
    int P=UserFindFirst();
    bool Found;
    if UserNameFound(P)!="" then Found=true; else Found=false;end
    While(Found==true)
        MessageBox(UserNameFound(P),UserGroupsFound(P));
        Found=UserFindNext(P);
    end
    UserFindClose(P);
end
```

9.13.9 UserFindClose

Description
It stops the process of user file reading (started with User*FindFirst*) then free used resources

Syntax
`void UserFindClose(int Handle)`

Parameters
Handle this is the handle of the search in progress.

Returned value
-
Related functions
UserFindFirst() , UserFindNext() , UserNameFound() ,UserGroupsFound()

Example

```
Function void FindUsers()
    int P=UserFindFirst();
    bool Found;
    if UserNameFound(P)!="" then Found=true; else Found=false;end
    While(Found==true)
        MessageBox(UserNameFound(P),UserGroupsFound(P));
        Found=UserFindNext(P);
    end
    UserFindClose(P);
end
```

9.13.10 UserGroupsFound

Description
It returns the groups belonging to the last user read by User*FindFirst or UserFindNext*

Syntax
`Int UserGroupsFound(int Handle)`

Parameters
Handle this is the handle of the search in progress.

Returned value
An integer value that represents in binary format the groups associated with the User.(Refer to GetUserGroups() for more details)

Related functions
UserFindFirst() , UserFindNext() , UserFindClose() , UserNameFound()

Example

```
Function void FindUsers()
    int P=UserFindFirst();
    bool Found;
    if UserNameFound(P)!="" then Found=true; else Found=false;end
    While(Found==true)
        MessageBox(UserNameFound(P),UserGroupsFound(P));
        Found=UserFindNext(P);
    end
    UserFindClose(P);
end
```

9.13.11 UserNameFound

Description
It returns the name of last user read by User*FindFirst or UserFindNext*

Syntax
`String UserNameFound(int Handle)`

Parameters
Handle this is the handle of the search in progress.

Returned value
name of the User

Related functions

UserFindFirst() , UserFindNext() , UserFindClose() , UserGroupsFound()

Example

```
Function void FindUsers()
    int P=UserFindFirst();
    bool Found;
    if UserNameFound(P)!="" then Found=true; else Found=false;end
    While(Found==true)
        MessageBox(UserNameFound(P),UserGroupsFound(P));
        Found=UserFindNext(P);
    end
    UserFindClose(P);
end
```

9.14 Recipes

9.14.1 RecipeCreate

Description
It creates a recipe according to the defined model and parameters.

Syntax

```
Bool RecipeCreate(String Model, String Recipe, Bool ErrorMsg)
```

Parameters

Model	recipe model (existing)
Recipe	name of the recipe you want to create
ErrMsg	true if you want to display a message error

Returned value
true (if the operation has been successful)
false (if not)

Related functions
RecipeImport(), RecipeExecute()

Example

```
RecipeIsReady=RecipeCreate("Recipes", "Week 32", false);
```

9.14.2 RecipeExecute

Description
It sends the values concerning the specified recipe to the various devices.

Syntax

```
Bool RecipeExecute(String Model, String Recipe, Bool CompletionMsg)
```

Parameters

Model	model whose values you want to change
Recipe	model values
ErrMsg	true if you want to display a message error

Returned value
true (if the operation has been successful)
false (if not)

Related functions

RecipeCreate(), RecipeImport()

Example
```
RecipeExecute (CurrModel,NewInsertedValues, true)
```

9.14.3 RecipeImport

Description
It saves the values concerning the model in the specified recipe.

Syntax
```
Bool RecipeImport(String Model, String Recipe, Bool CompletionMsg)
```

Parameters
Model model whose values you want to read (existing)
Recipe recipe you want to import the values to (existing)
ErrMsg true if you want to display a message error

Returned value
true (if the operation has been successful)
false (if not)

Related functions
RecipeCreate(), RecipeExecute()

Example
```
RecipeImport (CurrModel,CurrValues, false);
```

9.15 Report

9.15.1 ReportAppendRecord

Description
This command is used for DAT report configured to "Save record on command". When this command is executed,a new record will be appended to the present report file;if the file doesn't exist,it will be created.
If the report is configured to be saved in the default report directory, ReportCreate() function can be used to automatically create a new file.In this case the previous report file will be renamed from (ReportName).001 to (ReportName).002 and a new file will be created with the name (ReportName).001.

Syntax
void ReportAppendRecord(`String Name`)

Parameters
Name name of the report type (existing)

Returned value
-

Related functions
ReportCreate()

Example
```
ReportAppendRecord("ReportName");
```

9.15.2 ReportCreate

Description
It draws up a Report file.

Syntax
```
Bool ReportCreate(String Name)
```

Parameters
Name name of the report type (existing)

Returned value
true (if the operation has been successful)
false (if not)

Related functions
ReportDisplay(), ReportPrint() , ReportSetFullPathFileName()

Example
```
FileCreatedFlag= ReportCreate("CurrentSystemStatus");
```

9.15.3 ReportDisplay

Description
It displays the last created Report file.
This function is not available for .DAT report.

Syntax
```
Bool ReportDisplay(String Name)
```

Parameters
Name name of the report file

Returned value
true (if the operation has been successful)
false (if not)

Related functions
ReportCreate(), ReportPrint()

Example
```
FileSuccesfullyReaded= ReportDisplay("SystemStatus");
```

9.15.4 ReportGetPeriodType

Description:
Returns an integer indicating the report creation frequency

Syntax:
```
Int ReportGetPeriodType(String ReportName)
```

Parameters:
ReportName name of the report file

Returned value:
0 Never (report don't be created)

1 Time (report is created periodically at specified time interval)
2 DayOfWeek (report is created once a week)
3 DayOfMonth (report is created once a month)
4 DayAndMonth (report is created once a year)

Related functions:
ReportSetPeriodNone(),
ReportSetPeriodTime(),
ReportSetPeriodDayOfWeek(),
ReportSetPeriodDayOfMonth(),
ReportSetPeriodDayAndMonth()

Example:
```
GroupAReportTime = ReportGetPeriodTime(GroupAReportFileName);
```

9.15.5 ReportInsertText

Description
It insert text into a report during its creation.
This function must be used only inside a report TXT or RTF/PDF.

Syntax
```
void ReportInsertText(String text)
```

Parameters
text text that will be inserted into report

Returned value
-

Related functions
ReportInsertTemplate()

Example
ReportInsertText, ReportInsertTemplate example

Note: Restrictions

9.15.6 ReportInsertTemplate

Description
It insert template image into a report during its creation.
(Obsolete function - now replaced from ReportinsertTemplateEx)
This function must be used only inside a RTF/PDF report.

Syntax
```
void ReportInsertTemplate(String text)
```

Parameters
text template name which image will be inserted into report

Returned value
-

Related functions
ReportInsertText(),ReportInsertTemplateEx()

Example
ReportInsertText, ReportInsertTemplate example

Note: Template images are quite large, an image of a 1024x768 template is about 2.5Mbytes for a RTF report.

Restrictions

9.15.7 ReportInsertTemplateEx

Description
It insert template image into a report during its creation.
This function must be used only inside a RTF/PDF report .

Syntax
```
void ReportInsertTemplate(String text)
```

Parameters
text template name which image will be inserted into report

Returned value
-

Related functions
ReportInsertText()

Example
ReportInsertText, ReportInsertTemplate example

Note: Template images are quite large, an image of a 1024x768 template is about 2.5Mbytes for a RTF report.

Restrictions

9.15.8 ReportInsertHistoricalAlarmsRTF

Description
This function automatically insert an historical alarms / events list in a "RTF" report.

Syntax
```
void ReportInsertHistoricalAlarmsRTF(
                                int sDay,
                                int sMonth,
                                int sYear,
                                int sHour,
                                int sMin,
                                int sSec,
                                int eDay,
                                int eMonth,
                                int eYear,
                                int eHour,
                                int eMin,
                                int eSec,
                                int ListType,
```

```
bool ShowDescription,
string ColumnName1,
string ColumnName2,
string ColumnName3,
string ColumnName4,
string ColumnName5,
string ColumnName6,
string ColumnName7,
string ColumnName8,
string ColumnName9,
string ColumnName10,
string ColumnName11,
string ColumnName12,
string ColumnName13,
int FilterClass1,
string FilterClass2,
string FilterClass3,
string FilterClass4,
string FilterClass5,
string FilterClass6,
string FilterClass7)
```

PARAMETERS	DESCRIPTION
sDay = start day ***sMonth*** = start month ***sYear*** = start year ***sHour*** = start hour ***sMin*** = start minutes ***sSec*** = start seconds	Start date / Time from which start to insert in the report the historical alarms / events
eDay = end day ***eMonth*** = end month ***eYear*** = end year ***eHour*** = end hour ***eMin*** = end minutes ***eSec*** = end seconds	End date / Time to which stop to insert in report the historical alarms / events
ListType	0: show alarms 1: show events 2: show both alarms and events
ShowDescription	True : show header description of each configured column False: don't show header description
ColumName1 ***ColumName2*** ***ColumName3*** ***ColumName4*** ***ColumName5*** ***ColumName6*** ***ColumName7*** ***ColumName8*** ***ColumName9*** ***ColumName10*** ***ColumName11*** ***ColumName12*** ***ColumName13***	There are at maximum 13 columns. The type of data to show in each column is specified by *column name identifier* and can be one of the following name: "MESSAGE" : show the message of alarm/event "START_DATE": show start date of alarm/event "START_TIME": show start time of alarm/event "END_DATE": show end date of alarm/event "END_TIME": show end time of alarm/event "DURATION": show duration of alarm/event "CLASS1": show class1 "CLASS2": show class2

	"CLASS3": show class3 "CLASS4": show class4 "CLASS5": show class5 "CLASS6": show class6 "CLASS7": show class7 All the columns specified in ColumnName1..13 must be also specified in ProjectManager->Configuration->Template->HistoricalAlarms or HistoricalEvents. A NULL string (**""**) must be specified if the relative column must not be displayed.
FilterClass1 ***FilterClass2*** ***FilterClass3*** ***FilterClass4*** ***FilterClass5*** ***FilterClass6*** ***FilterClass7***	With these parameters is possible to specify a view filter for alarms/event in the report. *FilterClass1* is a number while *FilterClass2* to *FilterClass7* are string of char. *FilterClass1*= **-1** means don't care about Class1. (Show all alarms/events) *FilterClass2...FilterClass7*= **""** means don't care about Class2...Class7.

Returned value
None.

Remarks
How to set columns width?
Each column is separated by a TAB char, so in the source report file must be set the tabulator of each columns.
For example with *Microsoft WordPad,* tabulator can be set by selecting Format->Tabulator Menu Item.

Example

```
ReportInsertHistoricalAlarmsRTF(
                                10,11,2005,0,0,0,
                                10,11,2005,23,59,59,
                                0,
                                true,
                                "MESSAGE",
                                "START_DATE",
                                "START_TIME",
                                "END_DATE",
                                "END_TIME",
                                "DURATION",
                                "CLASS1",
                                "CLASS2",
                                "CLASS5",
                                "",
                                "",
                                "",
                                "",
                                -1,
                                "",
                                "",
                                "",
                                "",
                                "",
                                "");
```

9.15.9 ReportInsertHistoricalAlarmsTXT

Description
This function automatically insert an historical alarms / events list in a "TXT" report.

Syntax

```
void ReportInsertHistoricalAlarmsTXT(
                                     int sDay,
                                     int sMonth,
                                     int sYear,
                                     int sHour,
                                     int sMin,
                                     int sSec,
                                     int eDay,
                                     int eMonth,
                                     int eYear,
                                     int eHour,
                                     int eMin,
                                     int eSec,
                                     int ListType,
                                     bool ShowDescription,
                                     string ColumnName1,
                                     int ColumnWidth1,
                                     string ColumnName2,
                                     int ColumnWidth2,
                                     string ColumnName3,
                                     int ColumnWidth3,
                                     string ColumnName4,
                                     int ColumnWidth4,
                                     string ColumnName5,
                                     int ColumnWidth5,
                                     string ColumnName6,
                                     int ColumnWidth6,
                                     string ColumnName7,
                                     int ColumnWidth7,
                                     string ColumnName8,
                                     int ColumnWidth8,
                                     string ColumnName9,
                                     int ColumnWidth9,
                                     string ColumnName10,
                                     int ColumnWidth10,
                                     string ColumnName11,
                                     int ColumnWidth11,
                                     string ColumnName12,
                                     int ColumnWidth12,
                                     string ColumnName13,
                                     int ColumnWidth13,
                                     int FilterClass1,
                                     string FilterClass2,
                                     string FilterClass3,
                                     string FilterClass4,
                                     string FilterClass5,
                                     string FilterClass6,
                                     string FilterClass7)
```

PARAMETERS	DESCRIPTION

sDay = start day ***sMonth*** = start month ***sYear*** = start year ***sHour*** = start hour ***sMin*** = start minutes ***sSec*** = start seconds	Start date / Time from which start to insert in the report the historical alarms / events
eDay = end day ***eMonth*** = end month ***eYear*** = end year ***eHour*** = end hour ***eMin*** = end minutes ***eSec*** = end seconds	End date / Time to which stop to insert in report the historical alarms / events
ListType	0: show alarms 1: show events 2: show both alarms and events
ShowDescription	True : show header description of each configured column False: don't show header description
ColumName1, ***ColumnWidth1,*** ***ColumName2,*** ***ColumnWidth2,*** ***ColumName3,*** ***ColumnWidth3,*** ***ColumName4,*** ***ColumnWidth4,*** ***ColumName5,*** ***ColumnWidth5,*** ***ColumName6,*** ***ColumnWidth6,*** ***ColumName7,*** ***ColumnWidth7,*** ***ColumName8,*** ***ColumnWidth8,*** ***ColumName9,*** ***ColumnWidth9,*** ***ColumName10,*** ***ColumnWidth10,*** ***ColumName11,*** ***ColumnWidth11,*** ***ColumName12,***. ***ColumnWidth12,*** ***ColumName13,*** ***ColumnWidth13,***	There are at maximum 13 columns. The type of data to show in each column is specified by *column name identifier* and can be one of the following name: "MESSAGE" : show the message of alarm/event "START_DATE": show start date of alarm/event "START_TIME": show start time of alarm/event "END_DATE": show end date of alarm/event "END_TIME": show end time of alarm/event "DURATION": show duration of alarm/event "CLASS1": show class1 "CLASS2": show class2 "CLASS3": show class3 "CLASS4": show class4 "CLASS5": show class5 "CLASS6": show class6 "CLASS7": show class7 All the columns specified in ColumnName1..13 must be also specified in ProjectManager->Configuration->Template->HistoricalAlarms or HistoricalEvents A NULL string (**"")** must be specified if the relative column must not be displayed *Column Width identifier* specify the maximum number of chars for each column.
FilterClass1 ***FilterClass2*** ***FilterClass3*** ***FilterClass4*** ***FilterClass5*** ***FilterClass6*** ***FilterClass7***	With these parameters is possible to specify a view filter for alarms/event in the report. *FilterClass1* is a number while *FilterClass2* to *FilterClass7* are string of char. *FilterClass1*= **-1** means don't care about Class1. (Show all alarms/events) *FilterClass2...FilterClass7*= **""** means don't care about Class2...Class7.

Returned value
None.

Example
```
ReportInsertHistoricalAlarmsTXT(
                                  10,11,2005,0,0,0,
                                  10,11,2005,23,59,59,
                                  0,
                                  true,
                                  "MESSAGE",30,
                                  "START_DATE",13,
                                  "START_TIME",13,
                                  "END_DATE",13,
                                  "END_TIME",13,
                                  "DURATION",15,
                                  "CLASS1",3,
                                  "CLASS2",10,
                                  "CLASS5",10,
                                  "",0,
                                  "",0,
                                  "",0,
                                  "",0,
                                  -1,
                                  "",
                                  "",
                                  "",
                                  "",
                                  "",
                                  "");
```

9.15.10 ReportLotTime

Description:
Returns a string with the following format:
DD/MM/YYYY - HH:MN:SS

Syntax:
```
String ReportLotTime (
                       Int GG, Int MM, Int AAAA, Int HH, Int MN, Int SS)
```

Parameters:
GG day
MM month
AAAA year
HH hour
MN minutes
SS seconds

Returned value
formatted string with parameters

Related functions
-

Example:
```
String start;
start =  ReportLotTime (GetDayOfMonth(), GetMonth(), GetYear(),  0, 0, 0);
```

9.15.11 ReportInsertUserChangesRTF

Description
This function automatically insert an user changes list in a "RTF" report.

Syntax

```
void ReportInsertUserChangesRTF(
                                int sDay,
                                int sMonth,
                                int sYear,
                                int sHour,
                                int sMin,
                                int sSec,
                                int eDay,
                                int eMonth,
                                int eYear,
                                int eHour,
                                int eMin,
                                int eSec,
                                bool ShowDescription,
                                string ColumnName1,
                                string ColumnName2,
                                string ColumnName3,
                                string ColumnName4,
                                string ColumnName5)
```

PARAMETERS	DESCRIPTION
sDay = start day ***sMonth*** = start month ***sYear*** = start year ***sHour*** = start hour ***sMin*** = start minutes ***sSec*** = start seconds	Start date / Time from which start to insert in the report the user changes list
eDay = end day ***eMonth*** = end month ***eYear*** = end year ***eHour*** = end hour ***eMin*** = end minutes ***eSec*** = end seconds	End date / Time to which stop to insert in report the user changes list
ShowDescription	True : show header description of each configured column False: don't show header description
ColumName1 ***ColumName2*** ***ColumName3*** ***ColumName4*** ***ColumName5***	There are at maximum 5 columns. The type of data to show in each column is specified by *column name identifier* and can be one of the following name: "CODE" : show the operation code "USER": show the user that has done the operation "DATE": show the date of the operation "TIME": show the time of the operation "MESSAGE": show a description of the operation

	A NULL string (**""**) must be specified if the relative column must not be displayed.

Returned value
None.

Remarks
How to set columns width?
Each column is separated by a TAB char, so in the source report file must be set the tabulator of each columns.
For example with *Microsoft WordPad,* tabulator can be set by selecting Format->Tabulator Menu Item.

Example
```
ReportInsertUserChangesRTF(
                                10,11,2005,0,0,0,
                                10,11,2005,23,59,59,
                                true,
                                "CODE",
                                "USER",
                                "DATE",
                                "TIME",
                                "MESSAGE");
```

9.15.12 ReportInsertUserChangesTXT

Description
This function automatically insert an user changes list in a "TXT" report.

Syntax
```
void ReportInsertUserChangesTXT(
                                int sDay,
                                int sMonth,
                                int sYear,
                                int sHour,
                                int sMin,
                                int sSec,
                                int eDay,
                                int eMonth,
                                int eYear,
                                int eHour,
                                int eMin,
                                int eSec,
                                bool ShowDescription,
                                string ColumnName1,
                                int ColumnWidth1,
                                string ColumnName2,
                                int ColumnWidth2,
                                string ColumnName3,
                                int ColumnWidth3,
                                string ColumnName4,
                                int ColumnWidth4,
                                string ColumnName5,
                                int ColumnWidth5);
```

PARAMETERS	DESCRIPTION
sDay = start day ***sMonth*** = start month ***sYear*** = start year ***sHour*** = start hour ***sMin*** = start minutes ***sSec*** = start seconds	Start date / Time from which start to insert in the report the user changes list
eDay = end day ***eMonth*** = end month ***eYear*** = end year ***eHour*** = end hour ***eMin*** = end minutes ***eSec*** = end seconds	End date / Time to which stop to insert in report the user changes list
ShowDescription	True : show header description of each configured column False: don't show header description
ColumName1, ***ColumnWidth1,*** ***ColumName2,*** ***ColumnWidth2,*** ***ColumName3,*** ***ColumnWidth3,*** ***ColumName4,*** ***ColumnWidth4,*** ***ColumName5,*** ***ColumnWidth5,***	There are at maximum 5 columns. The type of data to show in each column is specified by *column name identifier* and can be one of the following name: "CODE" : show the operation code "USER": show the user that has done the operation "DATE": show the date of the operation "TIME": show the time of the operation "MESSAGE": show a description of the operation A NULL string (**""**) must be specified if the relative column must not be displayed *Column Width identifier* specify the maximum number of chars for each column.

Returned value
None.

Example

```
ReportInsertUserChangesTXT(
                                10,11,2005,0,0,0,
                                10,11,2005,23,59,59,
                                true,
                                "CODE",15,
                                "USER",12,
                                "DATE",10,
                                "TIME",10,
                                "MESSAGE",40);
```

9.15.13 ReportPrint

Description
It prints out the last created Report file.
This function is not available for .DAT report.

Syntax
```
Bool ReportPrint(String Name, Bool ViewPrintDlg)
```

Parameters
Name name of the report file

ViewPrintDlg
true (it displays the printer preferences before printing)
false (it doesn't)

Returned value
true (if the operation has been successful)
false (if not)

Related functions
ReportCreate(), ReportDisplay()

Example
```
FileSuccesfullyPrinted= ReportPrint("SystemStatus",true);
```

9.15.14 ReportSetFullPathFileName

Description
Specify the file name (with absolute full path) that must be associated with the next report that will be ceated.
This instruction is usually followed by ReportCreate()

Syntax
```
Bool ReportSetFullPathFileName(String ReportType,String FullPathFileName)
```

Parameters

ReportType	report type file name
FullPathFileName	name to be associated with the next report of type ReportType that will be created.The filename must be specified with the absolute complete path, like: "C:\DOCUMENTS\Report_07_10_1970.txt. If the path does not exist, ReportCreate() function does not create any report. If a file with the same name already exist it will be overwritten. If a file name has been specified with ReportSetFullPathFileName() function, all the reports that will be created after that function will overwrite the same file; to come back to the standard operation mode it is necessary to use again ReportSetFullPathFileName() function, specifing FullPathFileName="".

Returned value
true (if the operation has been successful)
false (if not)

Related functions
ReportCreate()

Example
```
 //=============================================================
 // Associate the name "C:\Documents\LotNumber1.rtf"  to the report type
"ProductionLot"
 //=============================================================
 if
(ReportSetFullPathFileName("ProductionLot","C:\Documents\LotNumber1.rtf")==
```

```
true) then

         //==================================
            // Create the report C:\Documents\LotNumber1.rtf
            //==================================
            ReportCreate("ProductionLot");

            //==================================
            // Return to the standart report operation mode
            //==================================
            ReportSetFullPathFileName("ProductionLot","");
end
```

9.15.15 ReportSetPeriodDayOfWeek

Description:
Specifies the day of the week when report has to be created.

Syntax:
```
Bool ReportSetPeriodDayOfWeek(String ReportName,Int Day,Int syncHour,Int
syncMin,Int syncSec)
```

Parameters:

ReportName	name of the report file
Day	day of week (1=mon, 2=tue, 3=wend...)
syncHour, syncMin, syncSec	start time

Returned value:
true (on success)
false (when file doesn't exist or specified time parameters are wrong)

Related fuctions:
ReportGetPeriodType(),
ReportSetPeriodNone(),
ReportSetPeriodTime(),
ReportSetPeriodDayOfMonth(),
ReportSetPeriodDayAndMonth()

Example:
```
ReportSetPeriodDayOfWeek(RepName,2,9,0,0); // tue by 9.00
```

9.15.16 ReportSetPeriodDayOfMonth

Description:
Specifies the day of the month when report has to be created.

Syntax:
```
Bool ReportSetPeriodDayOfMonth(String ReportNameInt Day,Int syncHour, Int
syncMin, Int syncSec)
```

Parameters:

ReportName	name of the report file
Day	day of month
syncHour, syncMin, syncSec	start time

Returned value:
true (on success)

false (when file doesn't exist or specified time parameters are wrong)

Related functions:
ReportGetPeriodType(),
ReportSetPeriodNone(),
ReportSetPeriodTime(),
ReportSetPeriodDayOfWeek(),
ReportSetPeriodDayAndMonth()

Example:
```
ReportSetPeriodDayOfMonth(RepName,24,9,0,0); // 24th by 9
```

9.15.17 ReportSetPeriodDayAndMonth

Description:
Specifies the day and month when report has to be created.

Syntax:
```
Bool ReportSetPeriodDayAndMonth(String ReportName, Int Day, int Month,Int
syncHour, Int syncMin, Int syncSec)
```

Parameters:

ReportName	name of the report file
Day	day
Month	month
syncHour, syncMin, syncSec	start time

Returned value:
true (on success)
false (when file doesn't exist or specified time parameters are wrong)

Related functions:
ReportGetPeriodType(),
ReportSetPeriodNone(),
ReportSetPeriodTime(),
ReportSetPeriodDayOfWeek(),
ReportSetPeriodDayOfMonth()

Example:
```
ReportSetPeriodDayOfMonth(RepName,24,1,9,0,0); // Jan  24th by 9.00
```

9.15.18 ReportSetPeriodNone

Description:
Specifies that the report must not be created

Syntax:
```
Bool ReportSetPeriodNone (String ReportName)
```

Parameters:

ReportName	name of the report file

Returned value:
true (on success)
false (on failure, may be that report doesn't exist)

Related functions:
ReportGetPeriodType(),

ReportSetPeriodTime(),
ReportSetPeriodDayOfWeek(),
ReportSetPeriodDayOfMonth(),
ReportSetPeriodDayAndMonth()

Example:
```
ReportSetPeriodNone(GroupAReportFileName);
```

9.15.19 ReportSetPeriodTime

Description:
Sets the report creation interval time

Syntax:
```
Bool ReportSetPeriodTime (String ReportName,Int OraPeriodo, Int MinPeriodo,
Int SecPeriodo,Int OraSincron, Int MinSincron,Int SecSincron)
```

Parameters:

ReportName	name of the report file
OraPeriodo, MinPeriodo, SecPeriodo	interval between reports
OraSincron, MinSincron, SecSincron	start time

Returned value:
true (on success)
false (when file doesn't exist or specified time parameters are wrong)

Related functions:
ReportGetPeriodType(),
ReportSetPeriodNone(),
ReportSetPeriodDayOfWeek(),
ReportSetPeriodDayOfMonth(),
ReportSetPeriodDayAndMonth()

Example:
```
ReportSetPeriodTime(RepName,1,0,0,9,0,0); // every hours by 9.00
```

9.15.20 ReportShowCreationList

Description
Shows the list of the historical reports to be created

Syntax
```
Void ReportShowCreationList()
```

Parameters
-

Returned value
-

Related functions
ReportShowHistList()

Example
```
ReportShowCreationList();
```

9.15.21 ReportShowHistList

Description
Shows the historical reports list.

Syntax
```
Void ReportShowHistList()
```

Parameters
-

Returned value
-

Related functions
ReportShowHistList()

Example
```
ReportShowHistList();
```

9.15.22 ReportInsertText, ReportInsertTemplate example

For example we have to create a daily report containing alarms found during a day.
Alarms will be listed in a text file which line will rappresent date and time of the event.

ALARMS.TXT:
```
....
1/2/2001 09:16:58
1/2/2001 09:19:26
1/2/2001 09:21:39
1/2/2001 09:21:59
1/2/2001 09:22:19
1/2/2001 09:22:40
1/2/2001 09:23:00
....
```

This file describe how report will be.
ALARMS.RTF:

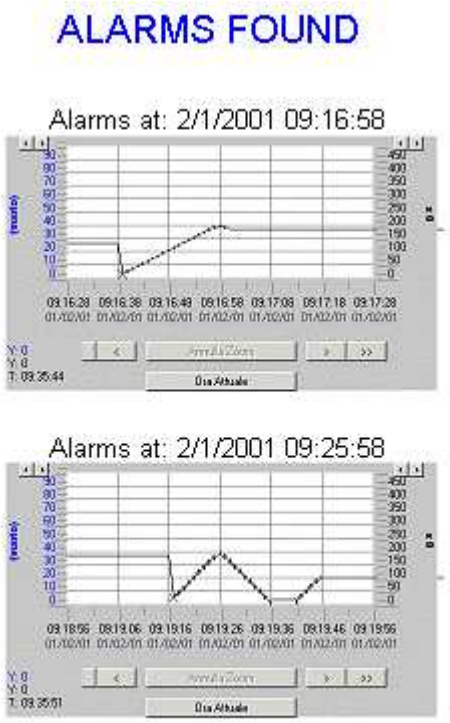
ALARMS FOUND:
{InsertAlarms()}

This is the function called inside the report:
ALARM.WLL:

```
Function void InsertAlarms()
 string line;
 int hh; int mn;  int ss; int yy; int mm; int dd;
 int h1; int m1; int s1; int h2; int m2; int s2;
 int a;
 int i=0;
 int h=FileOpen("E:\ALARMS:TXT","rt");
 While (FileEOF(h) == 0)
```

```
  line=FileReadLn(h);
  i=i+1;
  if (FileEOF(h) == 0) then
       ReportInsertText("Alarms at: "+line+Eol());
       // convert date and time from text to numerical values
       a=StrPos(line,"/"); dd=StrToInt(StrSubString(line,1,a-1)); line=StrDelete(line,1,a);
       a=StrPos(line,"/"); mm=StrToInt(StrSubString(line,1,a-1)); line=StrDelete(line,1,a);
       a=StrPos(line," "); yy=StrToInt(StrSubString(line,1,a-1)); line=StrDelete(line,1,a);
       a=StrPos(line,":"); hh=StrToInt(StrSubString(line,1,a-1)); line=StrDelete(line,1,a);
       a=StrPos(line,":"); mn=StrToInt(StrSubString(line,1,a-1)); line=StrDelete(line,1,a);
       // making a time range starting 30" before and ending 30" after
       ss=StrToInt(line);
       s1=ss-30; m1=mn; h1=hh;
       if (s1<0) then
           s1=60+s1;  m1=m1-1;
           if (m1==-1) then
               m1=59; h1=h1-1;
               if (h1<0) then
                   h1=23;
               end
           end
       end
       s2=s1;
       m2=m1+1;
       h2=h1;
       if (m2==60) then
               m2=0; h2=h2+1;
               if (h2==24) then
                   h2=0;
               end
       end
       ChartSetTimeRange(dd,mm,yy,h1,m1,s1,dd,mm,yy,h2,m2,s2); //da -30" a +30"
       // insert chart that show the problem found
       ReportInsertTemplate("AlarmChart");
       // template previously inserted AlarmChart
       // contains only one chart object that show the gates where problems were found
       ReportInsertText(Eol()+Eol()+Eol()+Eol());
  end
 end
 FileClose(h);
end
```

Resulting report at the end of the day can be like this **ALARMS.001**:



9.15.23 Report: Note

If you want to use the new report manipulation API:

```
ReportInsertText() and
ReportInsertTemplate()
```

you need the *riched32.dll* library in your system, this library is shipped with:

MS Windows NT 4.0 or newer version
MS Internet Explorer 4.0 or newer version
MS Office 97 or newer version

9.16 SMS

9.16.1 SMSCloseChannel

Description
Close communication with the specified GSM modem

Syntax

```
Bool SMSCloseChannel(Int Handle)
```

Parameters

Int Handle	handle associated with the serial port number connected to the GSM modem.

Returned value

True	operation completed successfully
False	error - probably the communication on the specified COM was not opened.

Related functions
SMSOpenChannel()

Example
```
Function void TestSignalQuality()
    int ComHandle = SMSOpenChannel(1,9600,"","");
    if (ComHandle>0) then
        MessageBox(SMSGetSignalQuality(ComHandle),"SMSSignalQuality");
        SMSCloseChannel(ComHandle);
    else
        MessageBox(ComHandle,"Error code");
    end
end
```

9.16.2 SMSDelete

Description
Delete the specified SMS from the SIM card

Syntax
```
Int SMSDelete(Int Handle, Int RecordIndex)
```

Parameters

Int Handle	handle associated with the serial port number connected to the GSM modem.
int RecordIndex	RecordIndex associated to the SMS that will be deleted.

Returned value

0	modem replies "OK"
-1	serial port can not be opened
-3	modem replies "ERROR"
-4	modem's reply is not a standard reply (it is not "OK" or "ERROR")
-5	modem does not reply

Related functions
SMSOpenChannel(), SMSCloseChannel(), SMSFindFirst()

Example
```
Function void TestSMSDelete()
    int ComHandle = SMSOpenChannel(1,9600,"","");
    if (ComHandle>0) then
        SMSDelete(comHandle,1);
        SMSCloseChannel(ComHandle);
    else
        MessageBox(ComHandle,"Error code");
    end
end
```

9.16.3 SMSFindClose

Description
Close the scanning of the received SMS list created with the instruction SMSFindFirst() and frees the memory.

Syntax
```
Bool SMSFindClose(Int Handle)
```

Parameters

Int Handle	handle associated with the serial port number connected to the GSM modem.

Returned value

True	Operation terminated successfully.
False	Invalid Handle has been specified.

Related functions

SMSOpenChannel(), SMSCloseChannel(), SMSFindFirst(), SMSFindNext(), SMSFoundRecordIndex() , SMSFoundTimeStamp(), SMSFoundTimeStampString(), SMSFoundRecordType(), SMSFoundSenderNumber(), SMSFoundSenderID(), SMSFoundMessage(),SMSFoundIsValid(), SMSDelete()

Example

```
Function void GetSMSList()
    int ComHandle = SMSOpenChannel(1,9600,"","");
    if (ComHandle>0) then
        if (SMSFindFirst(ComHandle,"ALL",true)==0) then
            ViewSMSData(ComHandle);
            while (SMSFindNext(ComHandle)==true) then
                ViewSMSData(ComHandle);
            end
            SMSFindClose(ComHandle);
        end
        SMSCloseChannel(ComHandle);
    else
        MessageBox(ComHandle,"Error code");
    end
end

Function void ViewSMSData(int ComHandle)
   MessageBox(SMSFoundRecordIndex(ComHandle),"RecodIndex");
   MessageBox(SMSFoundIsValid(ComHandle),"Is valid SMS ?");
   MessageBox(SMSFoundTimeStampString(ComHandle),"SMS TimeStampString");
   MessageBox(SMSFoundTimeStamp(ComHandle),"SMS TimeStamp");
   MessageBox(SMSFoundRecordType(ComHandle),"Record Type");
   MessageBox(SMSFoundSenderNumber(ComHandle),"Caller number");
   MessageBox(SMSFoundSenderID(ComHandle),"Caller ID");
   MessageBox(SMSFoundMessage(ComHandle),"SMS Message");
end
```

9.16.4 SMSFindFirst

Description

Read the received SMS list from the SIM card, sort in "Ascendant" or "Descendant" order, and points the first logical record.

Syntax

```
Int SMSFindFirst(Int Handle, Int Mode, Bool Sort)
```

Parameters

Int Handle	handle associated with the serial port number connected to the GSM modem.
Int Mode	can be one of the following case: 0 : received unread messages 1 : received read messages 2 : stored unsent message 3 : stored sent message 4 : all messages

Bool Sort	true: SMS list will be sorted in "Ascendant" order (from the older to the newer) false:SMS list wil be sorted in "Descendant" order (from the newer to the older)

Returned value

>0	Number of SMS records found. Fields of pointed SMS can be read using SMSFoundRecordIndex(), SMSFoundTimeStamp(), SMSFoundTimeStampString(), SMSFoundRecordType(), SMSFoundSenderNumber(), SMSFoundSenderID(), SMSFoundMessage(). Use SMSFindNext() to scan the SMS list. Use SMSFindClose() to terminating SMS list scanning.
0	modem replies "OK" but there are not SMS records found. No SMSFindClose() is needed.
-1	serial port can not be opened
-3	modem replies "ERROR"
-4	modem's reply is not a standard reply (it is not "OK" or "ERROR")
-5	modem does not reply

Related functions

SMSOpenChannel(), SMSCloseChannel(), SMSFindNext(), SMSFindClose(), SMSFoundRecordIndex(), SMSFoundTimeStamp(), SMSFoundTimeStampString(), SMSFoundRecordType(), SMSFoundSenderNumber(), SMSFoundSenderID(), SMSFoundMessage(), SMSFoundIsValid(),SMSDelete()

Example

```
Function void GetSMSList()
    int ComHandle = SMSOpenChannel(1,9600,"","");
    if (ComHandle>0) then
        if (SMSFindFirst(ComHandle,4,true)>0) then
            ViewSMSData(ComHandle);
            while (SMSFindNext(ComHandle)==true) then
                ViewSMSData(ComHandle);
            end
            SMSFindClose(ComHandle);
        end
        SMSCloseChannel(ComHandle);
    else
        MessageBox(ComHandle,"Error code");
    end
end

Function void ViewSMSData(int ComHandle)
   MessageBox(SMSFoundRecordIndex(ComHandle),"RecodIndex");
   MessageBox(SMSFoundIsValid(ComHandle),"Is valid SMS ?");
   MessageBox(SMSFoundTimeStampString(ComHandle),"SMS TimeStampString");
   MessageBox(SMSFoundTimeStamp(ComHandle),"SMS TimeStamp");
   MessageBox(SMSFoundRecordType(ComHandle),"Record Type");
   MessageBox(SMSFoundSenderNumber(ComHandle),"Caller number");
   MessageBox(SMSFoundSenderID(ComHandle),"Caller ID");
   MessageBox(SMSFoundMessage(ComHandle),"SMS Message");
end
```

9.16.5 SMSFindNext

Description

Move the pointer to the next logical SMS record of the received SMS list created with the instruction SMSFindFirst().

Syntax

```
Bool SMSFindNext(Int Handle)
```

Parameters

Int Handle handle associated with the serial port number connected to the GSM modem.

Returned value

True Next received SMS record found.
Fields of pointed SMS can be read using SMSFoundRecordIndex(), SMSFoundTimeStamp(), SMSFoundTimeStampString(), SMSFoundRecordType(), SMSFoundSenderNumber(), SMSFoundSenderID(), SMSFoundMessage()
Use SMSFindNext() to scan the SMS list.
Use SMSFindClose() to terminating SMS list scanning.

False End of received SMS list has been reached or invalid Handle has been specified.

Related functions

SMSOpenChannel(), SMSCloseChannel(), SMSFindFirst(), SMSFindClose(), SMSFoundRecordIndex(), SMSFoundTimeStamp(), SMSFoundTimeStampString(), SMSFoundRecordType(), SMSFoundSenderNumber(), SMSFoundSenderID(), SMSFoundMessage(), SMSFoundIsValid(),SMSDelete()

Example

```
Function void GetSMSList()
    int ComHandle = SMSOpenChannel(1,9600,"","");
    if (ComHandle>0) then
        if (SMSFindFirst(ComHandle,4,true)>0) then
            ViewSMSData(ComHandle);
            while (SMSFindNext(ComHandle)==true) then
                ViewSMSData(ComHandle);
            end
            SMSFindClose(ComHandle);
        end
        SMSCloseChannel(ComHandle);
    else
        MessageBox(ComHandle,"Error code");
    end
end

Function void ViewSMSData(int ComHandle)
   MessageBox(SMSFoundRecordIndex(ComHandle),"RecodIndex");
   MessageBox(SMSFoundIsValid(ComHandle),"Is valid SMS ?");
   MessageBox(SMSFoundTimeStampString(ComHandle),"SMS TimeStampString");
   MessageBox(SMSFoundTimeStamp(ComHandle),"SMS TimeStamp");
   MessageBox(SMSFoundRecordType(ComHandle),"Record Type");
   MessageBox(SMSFoundSenderNumber(ComHandle),"Caller number");
   MessageBox(SMSFoundSenderID(ComHandle),"Caller ID");
   MessageBox(SMSFoundMessage(ComHandle),"SMS Message");
end
```

9.16.6 SMSFoundIsValid

Description

Return true if the received SMS list pointed record is a supported SMS (i.e.: Simple text message with GSM 7bit default alphabet and SMS encoding).
Concatenated SMS are not supported.

Syntax

```
bool SMSFoundIsValid(Int Handle)
```

Parameters

Int Handle handle associated with the serial port number connected to the GSM modem.

Returned value

True SMS is supported
False SMS not supported

Related functions
SMSFindFirst(), SMSFindNext(), SMSFindClose(), SMSFoundRecordIndex(), SMSFoundTimeStamp() , SMSFoundTimeStampString(), SMSFoundRecordType(), SMSFoundSenderNumber(), SMSFoundSenderID()

Example

```
Function void GetSMSList()
    int ComHandle = SMSOpenChannel(1,9600,"","");
    if (ComHandle>0) then
        if (SMSFindFirst(ComHandle,4,true)>0) then
            ViewSMSData(ComHandle);
            while (SMSFindNext(ComHandle)==true) then
                ViewSMSData(ComHandle);
            end
            SMSFindClose(ComHandle);
        end
        SMSCloseChannel(ComHandle);
    else
        MessageBox(ComHandle,"Error code");
    end
end

Function void ViewSMSData(int ComHandle)
   MessageBox(SMSFoundRecordIndex(ComHandle),"RecodIndex");
   MessageBox(SMSFoundIsValid(ComHandle),"Is valid SMS ?");
   MessageBox(SMSFoundTimeStampString(ComHandle),"SMS TimeStampString");
   MessageBox(SMSFoundTimeStamp(ComHandle),"SMS TimeStamp");
   MessageBox(SMSFoundRecordType(ComHandle),"Record Type");
   MessageBox(SMSFoundSenderNumber(ComHandle),"Caller number");
   MessageBox(SMSFoundSenderID(ComHandle),"Caller ID");
   MessageBox(SMSFoundMessage(ComHandle),"SMS Message");
end
```

9.16.7 SMSFoundMessage

Description
Return the text message of the received SMS list pointed record.

Syntax
`String SMSFoundMessage(Int Handle)`

Parameters
Int Handle handle associated with the serial port number connected to the GSM modem.

Returned value
SMS message

Related functions
SMSFindFirst(), SMSFindNext(), SMSFindClose(), SMSFoundRecordIndex(), SMSFoundTimeStamp() , SMSFoundTimeStampString(), SMSFoundRecordType(), SMSFoundSenderNumber(), SMSFoundSenderID(),SMSFoundIsValid()

Example

```
Function void GetSMSList()
    int ComHandle = SMSOpenChannel(1,9600,"","");
    if (ComHandle>0) then
        if (SMSFindFirst(ComHandle,4,true)>0) then
            ViewSMSData(ComHandle);
            while (SMSFindNext(ComHandle)==true) then
                ViewSMSData(ComHandle);
            end
            SMSFindClose(ComHandle);
        end
        SMSCloseChannel(ComHandle);
    else
        MessageBox(ComHandle,"Error code");
    end
end

Function void ViewSMSData(int ComHandle)
   MessageBox(SMSFoundRecordIndex(ComHandle),"RecodIndex");
   MessageBox(SMSFoundIsValid(ComHandle),"Is valid SMS ?");
   MessageBox(SMSFoundTimeStampString(ComHandle),"SMS TimeStampString");
   MessageBox(SMSFoundTimeStamp(ComHandle),"SMS TimeStamp");
   MessageBox(SMSFoundRecordType(ComHandle),"Record Type");
   MessageBox(SMSFoundSenderNumber(ComHandle),"Caller number");
   MessageBox(SMSFoundSenderID(ComHandle),"Caller ID");
   MessageBox(SMSFoundMessage(ComHandle),"SMS Message");
end
```

9.16.8 SMSFoundRecordIndex

Description
Return the physical Record Index of the received SMS list pointed record.

Syntax
`Int SMSFoundRecordIndex(Int Handle)`

Parameters
Int Handle handle associated with the serial port number connected to the GSM modem.

Returned value
SMS Record Index

Related functions
SMSFindFirst(), SMSFindNext(), SMSFindClose(), SMSFoundTimeStamp(), SMSFoundTimeStampString(), SMSFoundRecordType(), SMSFoundSenderNumber(), SMSFoundSenderID(), SMSFoundMessage(),SMSFoundIsValid()

Example

```
Function void GetSMSList()
    int ComHandle = SMSOpenChannel(1,9600,"","");
    if (ComHandle>0) then
        if (SMSFindFirst(ComHandle,4,true)>0) then
            ViewSMSData(ComHandle);
            while (SMSFindNext(ComHandle)==true) then
                ViewSMSData(ComHandle);
            end
            SMSFindClose(ComHandle);
        end
        SMSCloseChannel(ComHandle);
    else
        MessageBox(ComHandle,"Error code");
    end
end
```

```
Function void ViewSMSData(int ComHandle)
   MessageBox(SMSFoundRecordIndex(ComHandle),"RecodIndex");
   MessageBox(SMSFoundIsValid(ComHandle),"Is valid SMS ?");
   MessageBox(SMSFoundTimeStampString(ComHandle),"SMS TimeStampString");
   MessageBox(SMSFoundTimeStamp(ComHandle),"SMS TimeStamp");
   MessageBox(SMSFoundRecordType(ComHandle),"Record Type");
   MessageBox(SMSFoundSenderNumber(ComHandle),"Caller number");
   MessageBox(SMSFoundSenderID(ComHandle),"Caller ID");
   MessageBox(SMSFoundMessage(ComHandle),"SMS Message");
end
```

9.16.9 SMSFoundRecordType

Description
Return the record type of the received SMS list pointed record..

Syntax
`String SMSFoundRecordType(Int Handle)`

Parameters
Int Handle handle associated with the serial port number connected to the GSM modem.

Returned value
"REC UNREAD" received unread message
"REC READ" received read message

Related functions
SMSFindFirst(), SMSFindNext(), SMSFindClose(), SMSFoundRecordIndex(), SMSFoundTimeStamp() , SMSFoundTimeStampString(), SMSFoundSenderNumber(), SMSFoundSenderID(), SMSFoundMessage(),SMSFoundIsValid()

Example

```
Function void GetSMSList()
    int ComHandle = SMSOpenChannel(1,9600,"","");
    if (ComHandle>0) then
        if (SMSFindFirst(ComHandle,4,true)>0) then
            ViewSMSData(ComHandle);
            while (SMSFindNext(ComHandle)==true) then
                ViewSMSData(ComHandle);
            end
            SMSFindClose(ComHandle);
        end
        SMSCloseChannel(ComHandle);
    else
        MessageBox(ComHandle,"Error code");
    end
end

Function void ViewSMSData(int ComHandle)
   MessageBox(SMSFoundRecordIndex(ComHandle),"RecodIndex");
   MessageBox(SMSFoundIsValid(ComHandle),"Is valid SMS ?");
   MessageBox(SMSFoundTimeStampString(ComHandle),"SMS TimeStampString");
   MessageBox(SMSFoundTimeStamp(ComHandle),"SMS TimeStamp");
   MessageBox(SMSFoundRecordType(ComHandle),"Record Type");
   MessageBox(SMSFoundSenderNumber(ComHandle),"Caller number");
   MessageBox(SMSFoundSenderID(ComHandle),"Caller ID");
   MessageBox(SMSFoundMessage(ComHandle),"SMS Message");
end
```

9.16.10 SMSFoundSenderID

Description
Return the caller ID of the received SMS list pointed record.

Syntax
`String SMSFoundSenderID(Int Handle)`

Parameters
Int Handle handle associated with the serial port number connected to the GSM modem.

Returned value
SMS Caller ID

Related functions
SMSFindFirst(), SMSFindNext(), SMSFindClose(), SMSFoundRecordIndex(), SMSFoundTimeStamp() , SMSFoundTimeStampString(), SMSFoundRecordType(), SMSFoundSenderNumber(), SMSFoundMessage(),SMSFoundIsValid()

Example
```
Function void GetSMSList()
    int ComHandle = SMSOpenChannel(1,9600,"","");
    if (ComHandle>0) then
        if (SMSFindFirst(ComHandle,4,true)>0) then
            ViewSMSData(ComHandle);
            while (SMSFindNext(ComHandle)==true) then
                ViewSMSData(ComHandle);
            end
            SMSFindClose(ComHandle);
        end
        SMSCloseChannel(ComHandle);
    else
        MessageBox(ComHandle,"Error code");
    end
end

Function void ViewSMSData(int ComHandle)
   MessageBox(SMSFoundRecordIndex(ComHandle),"RecodIndex");
   MessageBox(SMSFoundIsValid(ComHandle),"Is valid SMS ?");
   MessageBox(SMSFoundTimeStampString(ComHandle),"SMS TimeStampString");
   MessageBox(SMSFoundTimeStamp(ComHandle),"SMS TimeStamp");
   MessageBox(SMSFoundRecordType(ComHandle),"Record Type");
   MessageBox(SMSFoundSenderNumber(ComHandle),"Caller number");
   MessageBox(SMSFoundSenderID(ComHandle),"Caller ID");
   MessageBox(SMSFoundMessage(ComHandle),"SMS Message");
end
```

9.16.11 SMSFoundSenderNumber

Description
Return the caller number of the received SMS list pointed record.

Syntax
`String SMSFoundSenderNumber(Int Handle)`

Parameters
Int Handle handle associated with the serial port number connected to the GSM modem.

Returned value

SMS Caller number

Related functions
SMSFindFirst(), SMSFindNext(), SMSFindClose(), SMSFoundRecordIndex(), SMSFoundTimeStamp() , SMSFoundTimeStampString(), SMSFoundRecordType(), SMSFoundSenderID(), SMSFoundMessage(),SMSFoundIsValid()

Example
```
Function void GetSMSList()
    int ComHandle = SMSOpenChannel(1,9600,"","");
    if (ComHandle>0) then
        if (SMSFindFirst(ComHandle,4,true)>0) then
            ViewSMSData(ComHandle);
            while (SMSFindNext(ComHandle)==true) then
                ViewSMSData(ComHandle);
            end
            SMSFindClose(ComHandle);
        end
        SMSCloseChannel(ComHandle);
    else
        MessageBox(ComHandle,"Error code");
    end
end

Function void ViewSMSData(int ComHandle)
   MessageBox(SMSFoundRecordIndex(ComHandle),"RecodIndex");
   MessageBox(SMSFoundIsValid(ComHandle),"Is valid SMS ?");
   MessageBox(SMSFoundTimeStampString(ComHandle),"SMS TimeStampString");
   MessageBox(SMSFoundTimeStamp(ComHandle),"SMS TimeStamp");
   MessageBox(SMSFoundRecordType(ComHandle),"Record Type");
   MessageBox(SMSFoundSenderNumber(ComHandle),"Caller number");
   MessageBox(SMSFoundSenderID(ComHandle),"Caller ID");
   MessageBox(SMSFoundMessage(ComHandle),"SMS Message");
end
```

9.16.12 SMSFoundTimeStamp

Description
Return the time stamp of the received SMS list pointed record.

Syntax
`Unsigned SMSFoundTimeStamp(Int Handle)`

Parameters
Int Handle handle associated with the serial port number connected to the GSM modem.

Returned value
SMS time stamp expressed in number of seconds elapsed since January 1, 1901.

Related functions
SMSFindFirst(), SMSFindNext(), SMSFindClose(), SMSFoundRecordIndex(), SMSFoundTimeStampString(), SMSFoundRecordType(), SMSFoundSenderNumber(), SMSFoundSenderID(), SMSFoundMessage(),SMSFoundIsValid()

Example
```
Function void GetSMSList()
    int ComHandle = SMSOpenChannel(1,9600,"","");
    if (ComHandle>0) then
        if (SMSFindFirst(ComHandle,4,true)>0) then
```

```
            ViewSMSData(ComHandle);
            while (SMSFindNext(ComHandle)==true) then
                ViewSMSData(ComHandle);
            end
            SMSFindClose(ComHandle);
        end
        SMSCloseChannel(ComHandle);
    else
        MessageBox(ComHandle,"Error code");
    end
end

Function void ViewSMSData(int ComHandle)
   MessageBox(SMSFoundRecordIndex(ComHandle),"RecodIndex");
   MessageBox(SMSFoundIsValid(ComHandle),"Is valid SMS ?");
   MessageBox(SMSFoundTimeStampString(ComHandle),"SMS TimeStampString");
   MessageBox(SMSFoundTimeStamp(ComHandle),"SMS TimeStamp");
   MessageBox(SMSFoundRecordType(ComHandle),"Record Type");
   MessageBox(SMSFoundSenderNumber(ComHandle),"Caller number");
   MessageBox(SMSFoundSenderID(ComHandle),"Caller ID");
   MessageBox(SMSFoundMessage(ComHandle),"SMS Message");
end
```

9.16.13 SMSFoundTimeStampString

Description
Return the time stamp (string format) of the received SMS list pointed record.

Syntax
`String SMSFoundTimeStampString(Int Handle)`

Parameters
Int Handle handle associated with the serial port number connected to the GSM modem.

Returned value
SMS time stamp without any conversion.

Related functions
SMSFindFirst(), SMSFindNext(), SMSFindClose(), SMSFoundRecordIndex(), SMSFoundTimeStamp() , SMSFoundRecordType(), SMSFoundSenderNumber(), SMSFoundSenderID(), SMSFoundMessage() ,SMSFoundIsValid()

Example

```
Function void GetSMSList()
    int ComHandle = SMSOpenChannel(1,9600,"","");
    if (ComHandle>0) then
        if (SMSFindFirst(ComHandle,4,true)>0) then
            ViewSMSData(ComHandle);
            while (SMSFindNext(ComHandle)==true) then
                ViewSMSData(ComHandle);
            end
            SMSFindClose(ComHandle);
        end
        SMSCloseChannel(ComHandle);
    else
        MessageBox(ComHandle,"Error code");
    end
end

Function void ViewSMSData(int ComHandle)
   MessageBox(SMSFoundRecordIndex(ComHandle),"RecodIndex");
   MessageBox(SMSFoundIsValid(ComHandle),"Is valid SMS ?");
   MessageBox(SMSFoundTimeStampString(ComHandle),"SMS TimeStampString");
```

```
   MessageBox(SMSFoundTimeStamp(ComHandle),"SMS TimeStamp");
   MessageBox(SMSFoundRecordType(ComHandle),"Record Type");
   MessageBox(SMSFoundSenderNumber(ComHandle),"Caller number");
   MessageBox(SMSFoundSenderID(ComHandle),"Caller ID");
   MessageBox(SMSFoundMessage(ComHandle),"SMS Message");
end
```

9.16.14 SMSGetSignalQuality

Description
Returns received signal strength indication of the GSM modem.

Syntax
`Int SMSGetSignalQuality(Int Handle)`

Parameters

Int Handle	handle associated with the serial port number connected to the GSM modem.

Returned value

0	-113 dBm or less
1	-111 dBm
2..30	-109...-53 dBm
31	-51 dBm or greater
99	not know or not detectable
-1	serial port can not be opened
-3	modem replies "ERROR"
-4	modem's reply is not a standard reply (it is not "OK" or "ERROR")
-5	modem does not reply

Related functions
SMSOpenChannel(), SMSCloseChannel()

Example

```
Function void TestSignalQuality()
    int ComHandle = SMSOpenChannel(1,9600,"","");
    if (ComHandle>0) then
        MessageBox(SMSGetSignalQuality(ComHandle),"SMSSignalQuality");
        SMSCloseChannel(ComHandle);
    else
        MessageBox(ComHandle,"Error code");
    end
end
```

9.16.15 SMSOpenChannel

Description
Open communication with a GSM modem

Syntax
`Int SMSOpenChannel(Int COMx, Int Speed, String PIN, String ServiceCentreAddress)`

Parameters

Int COMx	serial port number associated to the GSM modem.
Int Speed	serial port communication speed (1200, 2400, 4800, 9600, 19200, 38400, 57600,115200).

String PIN	PIN number used to access the SIM. Specify "" to skip PIN setup.
String ServiceCentreAddress	number of the services center.It is possible to read the services center number on a mobile phone searching through menus.The same number can be requested to own telephonic service provider or it can be found in the SIM card guide.

Returned value

Handle	if it is a value >0 then it represents the handle associated with the opened COM
-1	serial port can not be opened
-3	modem replies "ERROR"
-4	modem's reply is not a standard reply (it is not "OK" or "ERROR")
-5	modem does not reply
-11	number of the service centre is too long (max 40 chars)

Related functions

SMSCloseChannel(), SMSSend(), SMSDelete(), SMSGetSignalQuality(), SMSFindFirst()

Example

```
Function void TestSignalQuality()
    int ComHandle = SMSOpenChannel(1,9600,"","");
    if (ComHandle>0) then
        MessageBox(SMSGetSignalQuality(ComHandle),"SMSSignalQuality");
        SMSCloseChannel(ComHandle);
    else
        MessageBox(ComHandle,"Error code");
    end
end
```

9.16.16 SMSSend

Description

Send an SMS to the specified destination number.
It is used the standard encoding for GSM messages with the 7 bit default alphabet GSM 03.38.
Concatenated SMS are not supported.
Maximum text length allowed is 160 characters.

Syntax

```
Int SMSSend(Int Handle, string Number, string Message)
```

Parameters

Int Handle	handle associated with the serial port number connected to the GSM modem.
String Number	phone number of the destination device
String Message	text message to send

Returned value

0	modem replies "OK"
-1	serial port can not be opened
-3	modem replies "ERROR"
-4	modem's reply is not a standard reply (it is not "OK" or "ERROR")
-5	modem does not reply
-11	destination number too long (max 40 chars)
-12	message too long (max 160 chars)

Related functions
SMSOpenChannel(), SMSCloseChannel(), SMSDelete(), SMSGetSignalQuality(), SMSFindFirst()

Example
```
Function void TestSMSSend()
    int ComHandle = SMSOpenChannel(1,9600,"","");
    if (ComHandle>0) then
        SMSSend(   comHandle,
                   GetStrGateValue("DestNumber",0),
                   GetStrGateValue("Message",0));
        SMSCloseChannel(ComHandle);
    else
        MessageBox(ComHandle,"Error code");
    end
end
```

9.17 String

9.17.1 CharToStr

Description
it makes a string of only one char

Syntax
`String CharToStr(Int  ASCIICode)`

Parameters
AsciiCode ASCII code of desired char

Returned value
string with the char

Related functions
Eol(), StrToReal(), StrToInt(), IntToStr(), RealToStr(), StrToChar()

Example
```
FGender = CharToStr(70);
```

9.17.2 Eol

Description
it returns the EndOfLine string

Syntax
`String Eol()`

Parameters
-

Returned value
string of one char (EOL char)

Related functions
CharToStr()

Example

```
Line = "first line!" + Eol() + "second line!";
```

9.17.3 IntToStr

Description
Converts an integer value into its string representation

Syntax
```
String IntToStr(Int Value)
```

Parameters
Value integer to conver

Returned value
string representation of the integer

Related functions
StrToInt(), StrToReal(), RealToStr(), ChatToStr()

Example
```
My=IntToStr(1975);
```

9.17.4 RealToStr

Description
Converts a real value into its string representation

Syntax
```
String RealToStr(Real Value)
```

Parameters
Value real value to convert

Returned value
String representation of the real value

Related functions
StrToInt(), StrToReal(), IntToStr(), ChatToStr()

Example
```
AbsZero = RealToStr(-273.5);
```

9.17.5 StrCase

Description
Converts a string to lower case or upper case.

Syntax
```
String StrCase(String S, Bool UpperCased)
```

Parameters
S string to process
UpperCased
true (convert to uppercase)
false (convert to lowercase)

Returned value
The converted string

Related functions
-

Example
```
LowCase = StrCase("VaLUe",false); // LowCase= "value"
```

9.17.6 StrConcat

Description
Concatenates two strings.

Syntax
```
String StrConcat(String S1, String S2)
```

Parameters

S1	first string
S2	second string

Returned value
The string result of the concatenation of the two strings.

Related functions
StrDelete()

Example
```
text = StrConcat ("Name","Spock");  // text = "NameSpock"
```

9.17.7 StrDelete

Description
Deletes a part (substring) of a string.

Syntax
```
String StrDelete(String S, Int Start, Int Len)
```

Parameters

S	string to process
Start	starting position of the substring to delete
Len	length of substring

Returned value
The string without the specified substring.

Related functions
StrSubString(), StrPos()

Example
```
Res = StrDelete("it is not raining",7,4); // Res = "it is raining"
```

9.17.8 StrLen

Description
it returns the length of a string

Syntax
```
Int StrLen(String Value)
```

Parameters
Value string to process

Returned value
the length of the string

Related functions
-

Example
```
NameLen = StrLen(Name);
```

9.17.9 StrOfChar

Description
Builds a string composed of the same char repeated.

Syntax
```
String StrOfChar(Int Chr, int Len)
```

Parameters
Chr char to be repeated
Len total length of string

Returned value
string created with specified parameters

Related functions
-

Example
```
TenB = StrOfChar (66,10);           // = "BBBBBBBBBB"
```

9.17.10 StrPos

Description
Searches for a substring in a string

Syntax
```
Int StrPos(String S, String SubStr)
```

Parameters
S string to search in
SubStr substring to search

Returned value
position of the substring (if found), otherwise 0

Related functions

StrDelete(), StrSubString()

Example
```
At = StrPos("it is not raining","not"); // At = 7
```

9.17.11 StrSubString

Description
Extracts a substring from a string.

Syntax
```
String StrSubString(String S, Int Start, Int Len)
```

Parameters

S	string to process
Start	starting position of the substring to extract
Len	length of the substring

Returned value
The substring specified by parameters.

Related functions
StrDelete(),StrPos()

Example
```
Res = StrSubString("it is not raining",7,4); // Res = "not "
```

9.17.12 StrToChar

Description
Returns a char contained in a string, specified by its position.

Syntax
```
Int StrToChar (String Str, Int Pos)
```

Parameters

Str	string to process
Pos	position of desired char (1 is the first char)

Returned value
The char of the string at the given position

Related functions
Eol(), StrToReal(), StrToInt(), IntToStr(), RealToStr(), CharToStr()

Example
```
Int Letter = StrToChar(InputField, i);
```

9.17.13 StrToInt

Description
it converts an integer value within a string to integer

Syntax
```
Int StrToInt(String Value)
```

Parameters

Value string that contains number

Valore resituito
integer value

Related functions
StrToReal(), IntToStr(), RealToStr(), CharToStr()

Example
```
Thickness = StrToInt(InputDialog("Thickness", "Line","10"));
```

9.17.14 StrToReal

Description
it converts real value within a string to real format

Syntax
```
Real StrToReal(String Value)
```

Parameters
Value string that contains value

Returned value
value converted to real

Related functions
RealToStr(), StrToInt(), CharToStr(), IntToStr()

Example
```
Size = StrToReal(strSize);
```

9.18 Template

9.18.1 TPageClose

Description
It closes the specified template page.

Syntax
```
void TPageClose(Integer Page)
```

Parameters
Page number of page to close

Returned value
-

Related functions
TPageOpen

Example
```
int genPage;
...
genPage = TPageOpen("Genesis");
MessageBox("Click to close","Genesis page");
TPageClose(genPage);
```

9.18.2 TPageCloseByName

Description
It closes all templates which name is the one specified as input at this function.

Syntax
```
void TPageCloseByName(string TemplateName)
```

Parameters
TemplateName name of the templates that must be closed

Returned value
-

Related functions
TPageOpen

Example
```
TPageOpen("Main");
MessageBox("Click to close","Main page");
TPageCloseByName("Main");
```

9.18.3 TPageOpen

Description
It opens a template page giving a specific name to it.

Syntax
```
int TPageOpen(String Name)
```

Parameters
Name name to give to the page

Returned value
page number >1 (on success)
0 (on failure)

Related functions
TPageClose,TPageCloseByName

Example
```
int mainPage;
mainPage=TPageOpen("Genoma Project");
```

9.18.4 TPagePrint

Description
Prints the template page that call this function

Syntax
Bool TPagePrint(Bool ViewPrintDlg)

Parameters
ViewPrintDlg if true shows the print dialog before printing

Returned value
true (on success)

false (on failure)

Related functions
-

Example
```
if (TPagePrint(true)) then
  MessageBox("Printed","RESULT");
else
  MessageBox("Print failed","RESULT");
end
```

9.18.5 TemplateAlarmsStatus

Description
Open the alarms status template.

Syntax
```
void TemplateAlarmsStatus()
```

Parameters
-

Returned value
-

Related functions
TemplatePrinterSetup() , TemplateRecipe() , TemplateSystemStatus() , TemplateDevicesStatus() , TemplateGatesStatus() , TemplateAlarmsStatus() , TemplateEventsStatus() , TemplateHistAlarms() , TemplateHistEvents() , TemplateUserChanges() , TemplateChart() , TemplateMultilanguage() , TemplateMakeReport() , TemplateViewReport() , TemplatePassword()

Example
```
TemplateAlarmsStatus();
```

9.18.6 TemplateChart

Description
Open the chart template.

Syntax
```
void TemplateChart()
```

Parameters
-

Returned value
-

Related functions
TemplatePrinterSetup() , TemplateRecipe() , TemplateSystemStatus() , TemplateDevicesStatus() , TemplateGatesStatus() , TemplateAlarmsStatus() , TemplateEventsStatus() , TemplateHistAlarms() , TemplateHistEvents() , TemplateUserChanges() , TemplateChart() , TemplateMultilanguage() , TemplateMakeReport() , TemplateViewReport() , TemplatePassword()

Example
```
TemplateChart();
```

9.18.7 TemplateDefineGroupNames

Description
It opens the define groups name dialog.

Syntax
```
void TemplateDefineGroupNames()
```

Parameters
-

Returned value
-

Related functions
TemplatePrinterSetup() , TemplateRecipe() , TemplateSystemStatus() , TemplateDevicesStatus() , TemplateGatesStatus() , TemplateAlarmsStatus() , TemplateEventsStatus() , TemplateHistAlarms() , TemplateHistEvents() , TemplateUserChanges() , TemplateChart() , TemplateMultilanguage() , TemplateMakeReport() , TemplateViewReport() , TemplatePassword(),TemplateDefineGroupNames()

Example
```
TemplateDefineGroupNames();
```

9.18.8 TemplateDefinePagesAccess

Description
It opens the define pages access dialog.

Syntax
```
void TemplateDefinePagesAccess()
```

Parameters
-

Returned value
-

Related functions
TemplatePrinterSetup() , TemplateRecipe() , TemplateSystemStatus() , TemplateDevicesStatus() , TemplateGatesStatus() , TemplateAlarmsStatus() , TemplateEventsStatus() , TemplateHistAlarms() , TemplateHistEvents() , TemplateUserChanges() , TemplateChart() , TemplateMultilanguage() , TemplateMakeReport() , TemplateViewReport() , TemplatePassword(),TemplateDefineGroupNames()

Example
```
TemplateDefinePagesAccess();
```

9.18.9 TemplateDefineUsers

Description
It open the define user dialog.

Syntax
```
void TemplateDefineUsers()
```

Parameters
-

Returned value
-

Related functions
TemplatePrinterSetup() , TemplateRecipe() , TemplateSystemStatus() , TemplateDevicesStatus() , TemplateGatesStatus() , TemplateAlarmsStatus() , TemplateEventsStatus() , TemplateHistAlarms() , TemplateHistEvents() , TemplateUserChanges() , TemplateChart() , TemplateMultilanguage() , TemplateMakeReport() , TemplateViewReport() , TemplatePassword()

Example
```
TemplateDefineUsers();
```

9.18.10 TemplateDevicesStatus

Description
Open the devices status template.

Syntax
```
void TemplateDevicesStatus()
```

Parameters
-

Returned value
-

Related functions
TemplatePrinterSetup() , TemplateRecipe() , TemplateSystemStatus() , TemplateDevicesStatus() , TemplateGatesStatus() , TemplateAlarmsStatus() , TemplateEventsStatus() , TemplateHistAlarms() , TemplateHistEvents() , TemplateUserChanges() , TemplateChart() , TemplateMultilanguage() , TemplateMakeReport() , TemplateViewReport() , TemplatePassword()

Example
```
TemplateDevicesStatus();
```

9.18.11 TemplateEventsStatus

Description
Open the events status template.

Syntax
```
void TemplateEventsStatus()
```

Parameters
-

Returned value
-

Related functions
TemplatePrinterSetup() , TemplateRecipe() , TemplateSystemStatus() , TemplateDevicesStatus() , TemplateGatesStatus() , TemplateAlarmsStatus() , TemplateEventsStatus() , TemplateHistAlarms() , TemplateHistEvents() , TemplateUserChanges() , TemplateChart() , TemplateMultilanguage() , TemplateMakeReport() , TemplateViewReport() , TemplatePassword()

Example

```
TemplateEventsStatus();
```

9.18.12 TemplateGatesStatus

Description
Open the gates status template.

Syntax
```
void TemplateGatesStatus()
```

Parameters
-

Returned value
-

Related functions
TemplatePrinterSetup() , TemplateRecipe() , TemplateSystemStatus() , TemplateDevicesStatus() , TemplateGatesStatus() , TemplateAlarmsStatus() , TemplateEventsStatus() , TemplateHistAlarms() , TemplateHistEvents() , TemplateUserChanges() , TemplateChart() , TemplateMultilanguage() , TemplateMakeReport() , TemplateViewReport() , TemplatePassword()

Example
```
TemplateGatesStatus();
```

9.18.13 TemplateHistAlarms

Description
Open the history of the alarms template.

Syntax
```
void TemplateHistAlarms()
```

Parameters
-

Returned value
-

Related functions
TemplatePrinterSetup() , TemplateRecipe() , TemplateSystemStatus() , TemplateDevicesStatus() , TemplateGatesStatus() , TemplateAlarmsStatus() , TemplateEventsStatus() , TemplateHistAlarms() , TemplateHistEvents() , TemplateUserChanges() , TemplateChart() , TemplateMultilanguage() , TemplateMakeReport() , TemplateViewReport() , TemplatePassword()

Example
```
TemplateHistAlarms();
```

9.18.14 TemplateHistEvents

Description
Open the history of the events template.

Syntax
```
void TemplateHistEvents()
```

Parameters
-

Returned value
-

Related functions
TemplatePrinterSetup() , TemplateRecipe() , TemplateSystemStatus() , TemplateDevicesStatus() , TemplateGatesStatus() , TemplateAlarmsStatus() , TemplateEventsStatus() , TemplateHistAlarms() , TemplateHistEvents() , TemplateUserChanges() , TemplateChart() , TemplateMultilanguage() , TemplateMakeReport() , TemplateViewReport() , TemplatePassword()

Example
```
TemplateHistEvents();
```

9.18.15 TemplateMakeReport

Description
Open the report generation template.

Syntax
```
void TemplateMakeReport()
```

Parameters
-

Returned value
-

Related functions
TemplatePrinterSetup() , TemplateRecipe() , TemplateSystemStatus() , TemplateDevicesStatus() , TemplateGatesStatus() , TemplateAlarmsStatus() , TemplateEventsStatus() , TemplateHistAlarms() , TemplateHistEvents() , TemplateUserChanges() , TemplateChart() , TemplateMultilanguage() , TemplateMakeReport() , TemplateViewReport() , TemplatePassword()

Example
```
TemplateMakeReport();
```

9.18.16 TemplateMultilanguage

Description
Open the multi language template.

Syntax
```
void TemplateMultilanguage()
```

Parameters
-

Returned value
-

Related functions
TemplatePrinterSetup() , TemplateRecipe() , TemplateSystemStatus() , TemplateDevicesStatus() , TemplateGatesStatus() , TemplateAlarmsStatus() , TemplateEventsStatus() , TemplateHistAlarms() , TemplateHistEvents() , TemplateUserChanges() , TemplateChart() , TemplateMultilanguage() , TemplateMakeReport() , TemplateViewReport() , TemplatePassword()

Example

```
TemplateMultilanguage();
```

9.18.17 TemplatePassword

Description
It open the password dialog.

Syntax
```
void TemplatePassword()
```

Parameters
-

Returned value
-

Related functions
TemplatePrinterSetup() , TemplateRecipe() , TemplateSystemStatus() , TemplateDevicesStatus() , TemplateGatesStatus() , TemplateAlarmsStatus() , TemplateEventsStatus() , TemplateHistAlarms() , TemplateHistEvents() , TemplateUserChanges() , TemplateChart() , TemplateMultilanguage() , TemplateMakeReport() , TemplateViewReport() , TemplatePassword()

Example
```
TemplatePassword();
```

9.18.18 TemplatePrinterSetup

Description
Open the printer setup template.

Syntax
```
void TemplatePrinterSetup()
```

Parameters
-

Returned value
-

Related functions
TemplatePrinterSetup() , TemplateRecipe() , TemplateSystemStatus() , TemplateDevicesStatus() , TemplateGatesStatus() , TemplateAlarmsStatus() , TemplateEventsStatus() , TemplateHistAlarms() , TemplateHistEvents() , TemplateUserChanges() , TemplateChart() , TemplateMultilanguage() , TemplateMakeReport() , TemplateViewReport() , TemplatePassword()

Example
```
TemplatePrinterSetup();
```

9.18.19 TemplateRecipe

Description
Open the recipe manager template.

Syntax
void TemplateRecipe()

Parameters
-

Returned value
-

Related functions
TemplatePrinterSetup() , TemplateRecipe() , TemplateSystemStatus() , TemplateDevicesStatus() , TemplateGatesStatus() , TemplateAlarmsStatus() , TemplateEventsStatus() , TemplateHistAlarms() , TemplateHistEvents() , TemplateUserChanges() , TemplateChart() , TemplateMultilanguage() , TemplateMakeReport() , TemplateViewReport() , TemplatePassword()

Example
```
TemplateRecipe();
```

9.18.20 TemplateSystemStatus

Description
Open the system status template.

Syntax
```
void TemplateSystemStatus()
```

Parameters
-

Returned value
-

Related functions
TemplatePrinterSetup() , TemplateRecipe() , TemplateSystemStatus() , TemplateDevicesStatus() , TemplateGatesStatus() , TemplateAlarmsStatus() , TemplateEventsStatus() , TemplateHistAlarms() , TemplateHistEvents() , TemplateUserChanges() , TemplateChart() , TemplateMultilanguage() , TemplateMakeReport() , TemplateViewReport() , TemplatePassword()

Example
```
TemplateSystemStatus();
```

9.18.21 TemplateUserChanges

Description
Open the user's operations template.

Syntax
```
void TemplateUserChanges()
```

Parameters
-

Returned value
-

Related functions
TemplatePrinterSetup() , TemplateRecipe() , TemplateSystemStatus() , TemplateDevicesStatus() , TemplateGatesStatus() , TemplateAlarmsStatus() , TemplateEventsStatus() , TemplateHistAlarms() , TemplateHistEvents() , TemplateUserChanges() , TemplateChart() , TemplateMultilanguage() , TemplateMakeReport() , TemplateViewReport() , TemplatePassword()

Example

```
TemplateUserChanges();
```

9.18.22 TemplateViewReport

Description
Open the report manager template.

Syntax
```
void TemplateViewReport()
```

Parameters
-

Returned value
-

Related functions
TemplatePrinterSetup() , TemplateRecipe() , TemplateSystemStatus() , TemplateDevicesStatus() , TemplateGatesStatus() , TemplateAlarmsStatus() , TemplateEventsStatus() , TemplateHistAlarms() , TemplateHistEvents() , TemplateUserChanges() , TemplateChart() , TemplateMultilanguage() , TemplateMakeReport() , TemplateViewReport() , TemplatePassword()

Example
```
TemplateViewReport();
```

9.19 Template objects

9.19.1 Generic

9.19.1.1 TObjEndUpdate

Description
Terminates the modify session of the object and update it.
When you want to set some object properties trough TObjSetPropertyBool(),TObjSetPropertyInt(), TObjSetPropertyReal(),TObjSetPropertyString(),TObjSetPropertyUnsigned() functions,it is necessary to call before the
TObjBeginUpdate() function to initialize the modify section area of the object.
After setting all needed properties, it is necessary to call the TObjEndUpdate() to update the object.

Syntax
Void TObjEndUpdate(Int Id)

Parameters
Id identifier of the object. It is the number set in "ID" property object trough Template Builder.

Returned value
-

Related functions
TObjSetPropertyBool(),TObjSetPropertyInt(),TObjSetPropertyReal(),TObjSetPropertyString(), TObjSetPropertyUnsigned(),TObjBeginUpdate()

Exampl
```
...
TObjBeginUpdate(100);
TObjSetPropertyReal(100,"ScaleMin",20);
TObjSetPropertyReal(100,"ScaleMax",80);
```

```
TObjEndUpdate(100);
...
```

9.19.1.2 TObjFunction

Description
Send a command to a specified object.
For example, for the Chart object there is the possibility to open some configuration windows (like chart set or time range configuration) or to simulate the forward ,rewind , reset zoom button .
In this way you can use your own button, label, or bitmap objects to manipulate the Chart object.

Syntax
Void TObjFunction(Int Id,int Function)

Parameters
Id identifier of the object. It is the number set in "ID" property object trough Template Builder.
Function command to perform (Refer to the object help in the Template Builder to know the list of commands that can be set through this function.)

Returned value
-

Related functions
-

Example

```
...
TObjFunction(1,3);
...
```

9.19.1.3 TObjGetH

Description
It returns the height of the specified object.

Syntax

```
Int TObjGetH(Int Id)
```

Parameters
Id identifier of the object

Returned value
Height of the object

Related functions
TObjGetW(), TObjSetSize()

Example

```
ObjHeight=TObjGetH(ObjID);
```

9.19.1.4 TobjGetLButtonDownXY

Description
It returns the mouse pointer X,Y coordinates in the moment of the last pressure of the left button, inside the specified object.

Syntax
Int TObjGetLButtonDownXY(Int Id)

Parameters
Id identifier of the object

Returned value
Value containing XY coordinates.
X coordinate=Valure & 0x0000FFFF
Y coordinate=Value / 65536

Related functions
TobjGetLButtonUpXY()

Example
```
int LButtonDnXY=TobjGetLButtonDownXY(2);
 int LButtonDnX=BitAnd(LButtonDnXY,65535);
 int LButtonDnY=LButtonDnXY/65536;
```

9.19.1.5 TobjGetLButtonUpXY

Description
It returns the mouse pointer X,Y coordinates in the moment of the last release of the left button, inside the specified object.

Syntax
Int TObjGetLButtonUpXY(Int Id)

Parameters
Id identifier of the object

Returned value
Value containing XY coordinates.
X coordinate=Valure & 0x0000FFFF
Y coordinate=Value / 65536

Related functions
TobjGetLButtonDownXY()

Example
```
int LButtonUpXY=TobjGetLButtonUpXY(2);
 int LButtonUpX=BitAnd(LButtonUpXY,65535);
 int LButtonUpY=LButtonUpXY/65536;
```

9.19.1.6 TObjGetPropertyInt

Description
Return an Integer property of the specified object.

Syntax
Int TObjGetPropertyInt(Int Id, string PropertyName)

Parameters

Id	object identifier. It is the number set in the "ID" property of the object (see Template Builder).
PropertyName	name of the property to read. Refer to the object help in the Template Builder to know the list of properties that can be read trough this function.

Returned value
Numeric value of the property

Related functions

Example

```
...
int Index;
Index=TObjGetPropertyInt(100,"ItemSelected");
...
```

9.19.1.7 TObjGetPropertyString

Description
Return an String property of the specified object.
If property does not exist then an empty string will be returned.

Syntax
String TObjGetPropertyString(Int Id, string PropertyName)

Parameters

Id	object identifier. It is the number set in the "ID" property of the object (see Template Builder).
PropertyName	name of the property to read. Refer to the object help in the Template Builder to know the list of properties that can be read trough this function.

Returned value
String value of the property

Related functions

Example

```
...
String Path;
Path=TObjGetPropertyString(100,"PathSelected");
...
```

9.19.1.8 TObjGetStatus

Description
It returns the status of the specified object.

Syntax

```
Int TObjGetStatus(Int Id)
```

Parameters
Id identifier of the object

Returned value
status of the object

Related functions
TObjSetStatus()

Example

```
CurrStatus=TObjGetStatus(CurrObject);
```

9.19.1.9 TObjGetText

Description
It returns the text of the specified object; of course this function is useless to the objects that don't have a text (e.g.: BitMap).

Syntax
String TObjGetText(Int Id)

Parameters
Id identifier of the object

Returned value
text of the object

Related functions
TObjSetText()

Example
```
Caption=TobjGetText(CurrObj);
```

9.19.1.10 TObjGetW

Description
It returns the width of the specified object.

Syntax
```
Int TObjGetW(Int Id)
```

Parameters
Id identifier of the object

Returned value
Width of the object

Related functions
TObjGetH(), TObjSetSize()

Example
```
ObjWidth=TObjGetW(ObjID);
```

9.19.1.11 TObjGetX

Description
It returns the abscissa of the specified object through its identifier.

Syntax
```
Int TObjGetX(Int Id)
```

Parameters
Id identifier of the object

Returned value
the abscissa of the specified object

Related functions
TObjSetXY(),TObjGetY()

Example
```
posX=TObjGetX(CurrObj);
```

9.19.1.12 TObjGetY

Description
It returns the ordinate of the specified object through its identifier.

Syntax
```
Int TObjGetY(Int Id)
```

Parameters
Id identifier of the object

Returned value
the ordinate of the specified object

Related functions
TObjSetXY(),TObjGetX()

Example
```
posY=TObjGetY(CurrObj);
```

9.19.1.13 TObjBeginUpdate

Description
Initialize properties updating session of the specified object.
When you want to set some object properties trough TObjSetPropertyBool(),TObjSetPropertyInt(), TObjSetPropertyReal(),TObjSetPropertyString(),TObjSetPropertyUnsigned() functions,it is necessary to call before the
TObjBeginUpdate() function to initialize the modify section area of the object.
After setting all needed properties, it is necessary to call the TObjEndUpdate() to update the object.

Syntax
Void TObjBeginUpdate(Int Id)

Parameters
Id identifier of the object. It is the number set in "ID" property object trough Template Builder.

Returned value
-

Related functions
TObjSetPropertyBool(),TObjSetPropertyInt(),TObjSetPropertyReal(),TObjSetPropertyString(), TObjSetPropertyUnsigned(),TObjEndUpdate()

Example
```
...
TObjBeginUpdate(100);
TObjSetPropertyReal(100,"ScaleMin",20);
TObjSetPropertyReal(100,"ScaleMax",80);
TObjEndUpdate(100);
...
```

9.19.1.14 TObjSetPropertyBool

Description
Modify a Bool property of the specified object.

Syntax
Void TObjSetPropertyBool(Int Id, string PropertyName,bool Value)

Parameters

Id	object identifier. It is the number set in the "ID" property of the object (see Template Builder).
PropertyName	name of the property to set. Refer to the object help in the Template Builder to know the list of properties that can be set trough this function.
Value	value to set.

Returned value
-

Related functions
TObjSetPropertyInt(),TObjSetPropertyReal(),TObjSetPropertyString(),TObjSetPropertyUnsigned(), TObjBeginUpdate(),TObjEndUpdate()

Example

```
...
TObjBeginUpdate(100);
TObjSetPropertyBool(100,PropertyName,true);
TObjEndUpdate(100);
...
```

9.19.1.15 TObjSetPropertyInt

Description
Modify an Integer property of the specified object.

Syntax
Void TObjSetPropertyInt(Int Id, string PropertyName,Int Value)

Parameters

Id	object identifier. It is the number set in the "ID" property of the object (see Template Builder).
PropertyName	name of the property to set. Refer to the object help in the Template Builder to know the list of properties that can be set trough this function.
Value	value to set.

Returned value
-

Related functions
TObjSetPropertyBool(),TObjSetPropertyReal(),TObjSetPropertyString(),TObjSetPropertyUnsigned(), TObjBeginUpdate(),TObjEndUpdate()

Example

```
...
TObjBeginUpdate(100);
TObjSetPropertyInt(100,"DecimalNumber",1);
TObjEndUpdate(100);
...
```

9.19.1.16 TObjSetPropertyReal

Description
Modify a Real property of the specified object.

Syntax
Void TObjSetPropertyReal(Int Id, string PropertyName,Int Value)

Parameters

Id	object identifier. It is the number set in the "ID" property of the object (see Template Builder).
PropertyName	name of the property to set. Refer to the object help in the Template Builder to know the list of properties that can be set trough this function.
Value	value to set.

Returned value
-

Related functions
TObjSetPropertyBool(),TObjSetPropertyInt(),TObjSetPropertyString(),TObjSetPropertyUnsigned(), TObjBeginUpdate(),TObjEndUpdate()

Example

```
...
TObjBeginUpdate(100);
TObjSetPropertyReal(100,"ScaleMin",20);
TObjSetPropertyReal(100,"ScaleMax",80);
TObjEndUpdate(100);
...
```

9.19.1.17 TObjSetPropertyString

Description
Modify a String property of the specified object.

Syntax
Void TObjSetPropertyString(Int Id, string PropertyName,String Value)

Parameters

Id	object identifier. It is the number set in the "ID" property of the object (see Template Builder).
PropertyName	name of the property to set. Refer to the object help in the Template Builder to know the list of properties that can be set trough this function.
Value	value to set.

Returned value
-

Related functions
TObjSetPropertyBool(),TObjSetPropertyInt(),TObjSetPropertyReal(),TObjSetPropertyUnsigned(), TObjBeginUpdate(),TObjEndUpdate()

Example

```
...
TObjBeginUpdate(100);
TObjSetPropertyString(100,"DescriptionText","Hello World");
TObjEndUpdate(100);
```

...

9.19.1.18 TObjSetPropertyUnsigned

Description
Modify an Unsigned Integer property of the specified object.

Syntax
Void TObjSetPropertyUnsigned(Int Id, string PropertyName,Unsigned int Value)

Parameters

Id	object identifier. It is the number set in the "ID" property of the object (see Template Builder).
PropertyName	name of the property to set. Refer to the object help in the Template Builder to know the list of properties that can be set trough this function.
Value	value to set.

Returned value
-

Related functions
TObjSetPropertyBool(),TObjSetPropertyInt(),TObjSetPropertyReal(),TObjSetPropertyString(), TObjBeginUpdate(),TObjEndUpdate()

Example

```
...
TObjBeginUpdate(100);
TObjSetPropertyUnsigned(100,PropertyName,1);
TObjEndUpdate(100);
...
```

9.19.1.19 TObjSetSize

Description
It changes the size of the specified object through its identifier.

Syntax

```
Void TObjSetSize(Int Id, Int w, Int h)
```

Paramentri

Id	identifier of the object
w	width
h	height

Returned value
-

Related functions
TObjGetW(),TObjGetH()

Example

```
TObjSetSize(CurrObj,50,50);
```

9.19.1.20 TObjSetStatus

Description
It changes the status of the specified object.

Syntax
```
Void TObjSetStatus(Int Id, Int status)
```

Parameters
Id identifier of the object
Status new status

Returned value
-

Related functions
TObjGetStatus()

Example
```
TObjSetStatus(ObjNotIdentified,0);
```

9.19.1.21 TObjSetText

Description
It changes the text of the specified object; of course this function is useless to the objects that don't have a text (e.g. BitMap).

Syntax
```
Void TObjSetText(Int Id, String text)
```

Parameters
Id identifier of the object
Text new text

Returned value
-

Related functions
TObjGetText()

Example
```
TObjSetText(NuovoObj,"New");
```

9.19.1.22 TObjSetXY

Description
It changes the position of the specified object through its identifier.

Syntax
```
Void TObjSetXY(Int Id, Int x, Int y)
```

Paramentri
Id identifier of the object
x new X-axis
y new Y-axis

Returned value
-

Related functions
TObjGetX(),TObjGetY()

Example
```
TObjSetXY(CurrObj, 100, 100);
```

9.19.2 Chart

9.19.2.1 ChartEndDrawingFlagReset

Description
Resets the end drawing flag.
When a template containing a chart object is opened, ChartEndDrawingFlag returns true only after the supervisor have drawn all charts.

Syntax
```
void ChartEndDrawingFlagReset()
```

Parameters
-

Returned value
-

Related functions
ChartEndDrawingFlagStatus()

Example
```
ChartEndDrawingFlagReset();
TOpenPage("ChartContainer");
while ( ChartEndDrawingFlagStatus() == false )
end
```

9.19.2.2 ChartEndDrawingFlagStatus

Description
Returns the status of the End Drawing Flag.
When a template containing a chart object is opened, ChartEndDrawingFlag returns true only after the supervisor have drawn all charts.

Syntax
```
Bool ChartEndDrawingFlagStatus()
```

Parameters
-

Returned value
True (if chart is drawn)
False (otherwise)

Related functions
ChartEndDrawingFlagReset()

Example
```
ChartEndDrawingFlagReset();
TOpenPage("ChartContainer");
while ( ChartEndDrawingFlagStatus() == false )
end
```

9.19.2.3 ChartSetTimeRange

Description
Sets **start date time and** end **date time** parameters of a template chart contained in a template. The time range property of Chart object has to be *External*, *External start* or *External stop.*

Syntax
```
void ChartSetTimeRange(
                        int sDay,int sMonth,int sYear,
                        int sHour,int sMin,int sSec,
                        int eDay,int eMonth,int eYear,
                        int eHour,int eMin,int eSec)
```

Parameters

sDay	start day
sMonth	start month
sYear	start year
sHour	start hour
sMin	start minutes
sSec	start seconds
eDay	end day
eMonth	end month
eYear	end year
eHour	end hour
eMin	end minutes
eSec	end seconds

Returned value
-

Related functions
ChartSetTimeRangeStartWidth(),ChartSetTimeRangeEndWidth()
-

Example
```
ChartSetTimeRange (24,1,1975,0,0,0 ,24,1,2000,0,0,0);
```

9.19.2.4 ChartSetTimeRangeStartWidth

Description
Sets the **start date time** and **time range** parameters of a template chart contained in a template. The time range property of Chart object has to be *External*, *External start* or *External stop.*

Syntax
```
void ChartSetTimeRangeStartWidth(
                                  int sDay,int sMonth,int sYear,
                                  int sHour,int sMin,int sSec,
                                  int Width)
```

Parameters

sDay	start day
sMonth	start month
sYear	start year
sHour	start hour
sMin	start minutes
sSec	start seconds
Width	Time range (sec)

Returned value
-

Related functions
ChartSetTimeRange(),ChartSetTimeRangeEndWidth()

Example
```
ChartSetTimeRangeStartWidth (24,1,1975,0,0,0 ,3600);
```

9.19.2.5 ChartSetTimeRangeEndWidth

Description
Sets the **end date time** and **time range** parameters of a template chart contained in a template.
The time range property of Chart object has to be *External*, *External start* or *External stop.*

Syntax
```
void ChartSetTimeRangeEndWidth(
                                int sDay,int sMonth,int sYear,
                                int sHour,int sMin,int sSec,
                                int Width)
```

Parameters

sDay	end day
sMonth	end month
sYear	end year
sHour	end hour
sMin	end minutes
sSec	end seconds
Width	Time range (sec)

Returned value
-

Related functions
ChartSetTimeRange(),ChartSetTimeRangeStartWidth()
-

Example
```
ChartSetTimeRangeEndWidth (24,1,1975,0,0,0 ,3600);
```

9.19.3 HistAlarmsView

9.19.3.1 HisViewEnablePrintOnCreation

Description
Automatically prints the alarms historical view or events historical view after the template is opened.

Syntax
```
void HisViewEnablePrintOnCreation()
```

Parameters
-

Returned value
-

Related functions

-

Example
```
HisViewEnablePrintOnCreation();
```

9.19.3.2 HisViewSetTimeRange

Description
Sets the time range of an historical view contained in a template.
The Time range property of HistView object has to be *External*, *External start* or *External stop.*

Syntax
```
void HisViewSetTimeRange(
                            int sDay, int sMonth, int sYear,
                            int sHour, int sMin, int sSec,
                            int eDay, int eMonth, int eYear,
                            int eHour, int eMin, int eSec)
```

Parameters

sDay	start day
sMonth	start month
sYear	start year
sHour	start hour
sMin	start minutes
sSec	start seconds
eDay	end day
eMonth	end month
eYear	end year
eHour	end hour
eMin	end minutes
eSec	end seconds

Returned value
-

Related functions
HisViewSetTimeRangeStartWidth(),HisViewSetTimeRangeEndWidth()

Example
```
HisViewSetTimeRange (24,1,1975,0,0,0 ,24,1,2000,0,0,0);
```

9.19.3.3 HisViewSetTimeRangeStartWidth

Description
Sets the **start date time** and **time range** parameters of a Historical Alarms object contained in a template.
The time range property of Historical Alarms object has to be *External*, *External start* or *External stop.*

Syntax
```
void HisViewSetTimeRangeStartWidth(
                                    int sDay,int sMonth,int sYear,
                                    int sHour,int sMin,int sSec,
                                    int Width)
```

Parameters

sDay	start day
sMonth	start month
sYear	start year

sHour	start hour
sMin	start minutes
sSec	start seconds
Width	Time range (sec)

Returned value
-

Related functions
HisViewSetTimeRange(),HisViewSetTimeRangeEndWidth()

Example
```
HisViewSetTimeRangeStartWidth(24,1,1975,0,0,0 ,3600);
```

9.19.3.4 HisViewSetTimeRangeEndWidth

Description
Sets the **end date time** and **time range** parameters of a Historical alarms object contained in a template.
The time range property of thet object has to be *External*, *External start* or *External stop.*

Syntax
```
void HisViewSetTimeRangeEndWidth(
                                  int sDay,int sMonth,int sYear,
                                  int sHour,int sMin,int sSec,
                                  int Width)
```

Parameters

sDay	end day
sMonth	end month
sYear	end year
sHour	end hour
sMin	end minutes
sSec	end seconds
Width	Time range (sec)

Returned value
-

Related functions
HisViewSetTimeRange(),HisViewSetTimeRangeStartWidth()

Example
```
HisViewSetTimeRangeEndWidth(24,1,1975,0,0,0 ,3600);
```

10 Language Elements

10.1 Introduction

The language is made up of basic elements (letters, numbers, punctuation, etc.), semantic rules and a syntax that specify the correctness of the commands that can be created.

The *instructions* are expressions involving some variables and functions thanks to operators provided by the language itself.

The *variables* are memory areas where you can store the needed data. They are called variables because they can have various values.

There are several categories or *types* of variables. Their difference depends on the information they represent and on how much memory they take up.

The *functions* are set of instructions that are grouped together, in order to avoid writing them again, and because so they represent a kind of instruction more complex.

This way it is possible to 'enrich' the language with a sort of new instructions.

The *operators* are the same you can find in elementary mathematics with the addition of some new ones.

To simplify the writing of the code in a certain language there are some structures such as conditions, cycles, conditioned cycles etc.

The *conditions* are blocks of instructions carried out only if the condition actually takes place.

The *cycles* are blocks of instructions carried out a defined number of times.

The *conditioned cycles* are blocks of instructions carried out until the condition takes place.

The *expressions* let you create complex elements using simple ones.

All these instructions are written in text files that the interpreter is going to decode and execute.

10.2 Foreword

The language files interpreted by *RunTime* have the extension WLL (e.g. Filatoio2.wll).

For the language interpreter there's no difference between capitals and small letters (this is the so-called case insensitive):

(the double slash // specifies the beginning of a comment)

```
Real Value;    // declaration of the real variable 'value'

Real value;
```

The interpreter reading this code fragment will spot an error because he will notice that the element `value` has been defined twice.

Later on you will understand better the meaning of 'to define' and that 'element' is only a generic term that in the specific cases can mean 'variable', 'function', 'constant' etc.

It is important to separate the elements with spaces or something, otherwise they will be considered as a single unit.

```
realValue;
```

If you wanted to declare the variable Value you have made a mistake: the interpreter sees only one element, that is to say `realvalue`.

In every file can be defined global functions and variables visible to all the defined functions. The global variables are defined at the beginning of the file. The visibility is the possibility to access a datum; if a variable is not visible, then it can't be used at all.

10.3 Variables

They are elements containing the values to manipulate during the execution of a program or a function.
In the language there are two types of variables:

- **Global** and so visible to all the program functions even if they are in different files (global visibility)

Syntax
`Global Type Name [= Default value]`
`Type` is the type of variable.
`Name` is the name you want to give to the variable.

Example
```
Global Real ExactTemperature;
Global Int AbsoluteZero=-273;
```

- **Local**, that is to say defined in each function. They can be used only within the function context (local visibility)

Syntax
`Type Name [= Default value]`
`Type` is the type of variable.
`Name` is the name you want to give to the variable.

Example
```
Function Int AbsoluteTemperature(integer T0)
   Real TemperaturePT01;
   ...
End
```

The variable Temperature PT01 is local to the function AbsoluteTemperature and so can't be used outside it. Anyway you can use the same name for another local variable in another function.

10.4 Types

10.4.1 Introduction

The variables contain different information and, depending on the information they contain, they are divided into categories or types; there are different types of variable:

`Int`	32 bit integer signed values
Unsigned	32 bit unsigned values
`Real`	floating point values
`Bool`	boolean values: they can only be either `true` or `false`
`String`	strings of characters varying in length.

10.4.2 Int

They are 32 bit integer values with sign and can assume values from -2147483648 to +2147483647: values outside those limits will be truncated to the first low meanings 32 bit.

Example
```
Int Counter = -2147483648;
Int Difference = +2147483647;
```

10.4.3 Unsigned

They are 32 bit unsigned integer values and can assume values from 0 to 4294967295 : values outside those limits will be truncated to the first low meanings 32 bit.

Example
```
Int Counter = 0;
Int Difference = 4294967295;
```

10.4.4 Real

They are real value with floating point that can store number with high precision

Example
```
Real pi;
Pi=3.141592;
```

10.4.5 Boolean

They are boolean values so they can have only two states **true** or **false**.

Example
```
Bool LedState;
LedState=GetDigGateValue("LEDPORT",0);
```

10.4.6 String

they are string of char with different length.

Example
```
String Name = "September";
```

10.5 Functions

The function is a procedure that is a list of instructions written by the application developer, selected to perform a specific task, such as calculating the result of a mathematical formula, read or write a file from / to disk, display data in a certain format, send commands to devices connected with the application, generate production reports and countless other tasks.

The functions are then of macroinstructions created by the same developer according to his needs. An advantage of the use of the functions is to avoid code redundancy: if I have to calculate the area of 10 different rectangle it is not necessary to rewrite the area of rectangle formula 10 times, but can define a function that will receive two input parameters ie Base and Height and return as result the calculation of the area of rectangle: the formula for the calculation shall be written once in the function and at this point to calculate the area of the 10 rectangles I simply call the function just defined 10 times, passing each time the parameters for the rectangle of which I plan to calculate the area.

A function is characterized by:
a return value which can be either one of the variable types supported by the software (types) or void if the function should not return anything.
a name *(case insensitive)* that identifies the function throughout the program
a list of parameters *(optional)* . They are the values the function needs in order to be operative.
directives *(optional).* They are special commands that enrich the functions
local variables *(optional),* necessary for the function development
a sequence of instructions characterizing the function.

It's better to give the functions meaningful names in order to simplify work. So if a function deals with the calculation of the temperature in Kelvin degrees, then you can call it 'CalculateTemperatureK'.

Syntax:

```
Function Type Name([Parameter List)
   [Directives]
   [Local Vars]
   Instructions
End
```

Type is the type of information returned by the function.(types)
Name is the name identifying the function.
Parameter List are the list and the types of data to be entered in the function so that it can perform its task.
Local Vars is the definition of all the variables necessary for the function development.
Instructions is the sequence of instructions carried out by the function.

The directives indicate:
#Macro: the name of the function is inserted in the **macro** menu of *RunTime*; so the function can be called up by the user.
#Startup: the function is executed at the start of *RunTime* in background. So it is possible to create cyclic functions performing checks or actions determined by certain conditions.
#Shutdown: the function is executed when you close *RunTime*.
#Modal: *RunTime* waits for the execution of the function to end (this mode keeps the operator from intervening but it doesn't stop the sampling and the registration of the gates).

Example:

```
Function Void MainCycle()
#Startup
      int in1;
      int in2;
      while (WindowIsOpen())

            in1 = GetDigGateValue("101IN",1);
            in2 = GetDigGateValue("101IN",2);

            if (in1+in2 == 2) then
                  SetDigGateValue("103OUT",1,1);
            else
                  SetDigGateValue("103OUT",1,0);
            end

            in1 = GetDigGateValue("101IN",48);
            in2 = GetDigGateValue("101IN",49);

            if (in1+in2 > 0) then
                  SetDigGateValue("103OUT",2,1);
            else
                  SetDigGateValue("103OUT",2,0);
            end

            if (Abs(GetCmpGateValue("Tr-Ts",1)) >GetNumGateValue("102DeltaT",1)) then
                  SetDigGateValue("103OUT",3,1);
            else
                  SetDigGateValue("103OUT",3,0);
            end
            Sleep(100);
      end
End
```

Putting properly into columns the instructions allows a better reading and understanding of a function.

You can call up functions when you open or close a *Template,* or by using commands (e.g. a *Button*) defined in the *Template* itself.

Notes : You can't execute a function if it is already being executed.

10.6 Instructions

10.6.1 Introduction

Every instruction must end with '**;**'. Let's see now the various instructions and structures the language is made up of.

Reserved words like **end, if, while** etc don't need '**;**' .

10.6.2 Function call

They are used if you need to call a function

Syntax
FunctionName(*[Parameter Expressions]*);

Example
```
PompaSpenta = PompaAttiva() - 3;
AttivaPompaNum(3);
```

10.6.3 Assignment

It's used when you want di give a new value to a variable.

Syntax
VarName = *Expression*;

Example
```
NumeroDiPID = (( 9 * Var1 ) / ( 1 - Var3 )) * ( 2 - Var2 );
Temp = TriggerValue - CurrValue;
```

10.6.4 Return value

This reserved word is used when you need to exit from current function and get back to caller function; by this you can return a value to caller function.

Syntax
Return *Expression*;

Example
```
Function void Caller()
   ...
   Int A;
   ..
   ASqr = Quadrato(A);
   ...
   End


Function int Quadrato(int Numero)
   int Quad;
   Quad = Numero * Numero;
   Return (Quad);
End
```

10.6.5 If Then Else condition

If it is used in the form:
If () Then
...
End
it executes the block of statements between *Then* and *End* if the condition specified in the *If()* statement is verified.

If it is used in the form:
If () Then
...
Else
...
End
it executes block of instructions between *Then* and *Else* if the condition specified in the *If()* statement is verified, otherwise it executes the block of instructions specified between *Else* and *End*.

Syntax
If (*Condition*) Then *Instructions*
*[*Else *Instructions]*
End

Example:
```
If (ValvolaAperta == 1) Then
   ChiudiValvolaPrimaria();
Else
   ChiudiValvolaNum(ValvolaAperta);
End

If (Temperatura < 75) Then
   Giri = 2000 - Temperatura * 5;
   SelezionaGiriMotoreNum(5,Giri);
End
```

10.6.6 For Next cycle

Repeat for a defined number of times the statements block between *For* and *End*.
It needs an integer value as counter.

Syntax
For *VarName* = *Expression* To *Expression* Do
  [Instructions]
End

Example
```
For PumpNumber = 2 To TotalPump Do
   SelectPump(PumpNumber);
   Level = CheckLevel();
   ActivePump(Level / 10);
End
```

Note:
Code has high priority inside the application; building a long duration loop (as can be done with "For" statement) can slow down the application response time: to avoid this, it is better to insert a "Sleep()" instruction inside the loop in order to let the system to handle the rest of the application more efficiently (i.e. window refresh,etc.).

10.6.7 While cycle

If the condition specified in the *While* brakets () is verified, executes the block of statements between *While()* and *end* statemets, then returns to test the condition in *While ()* and repeats the loop until the condition no longer be verified .

Syntax
While (*Comparison*)
  [Instructions]
End

Example
```
While (Temperature < 100)
   Speed = Speed + 1;
   ReadTemperature(Temperature);
   Sleep(100);
End
```

Note:
Code has high priority inside the application; building an infinite loop or a long duration loop (as can be done with "While" statement) can slow down the application response time: to avoid this, it is better to insert a "Sleep()" instruction inside the loop in order to let the system to handle the rest of the application more efficiently (i.e. window refresh,etc.).

10.6.8 Do While cycle

Executes the block of statements between *Do* and *While ()* at least one time, and if the condition specified in the *While* brakets () is verified then return to *Do* statement and repeats the operation again.
It remains in this loop until the condition in *While()* will no longer be verified.

Syntax
Do
  [Instructions]
While (*Comparison*)

Example
```
i = 0;
Do
   LightOn(i);
   i = i + 1;
While( i < 10)
```

Note:
Code has high priority inside the application; building an an infinite loop or a long duration loop (as can be done with "Do - While" statement) can slow down the application response time: to avoid this, it is better to insert a "Sleep()" instruction inside the loop in order to let the system to handle the rest of the application more efficiently (i.e. window refresh,etc.).

10.7 Expressions

The expressions are the elements the language is based on.
An expression can be a constant, a variable or a complex equation. It doesn't perform an action but it is simply evaluated.

As in the mathematical expressions the elements are linked by logic and mathematical operators.
The result of an expression is a value that in its turn can be used as an element of another expression; this way it is possible to put together even very complex formulas.
Here are some examples of expressions:

EXAMPLES OF EXPRESSION	DESCRIPTION
3	a numerical constant
ValorePortaA	a variable
log(14)	a function returning a value
4 + ValorePortaB	the addition operator is there
PortaAon && PortaBon	the boolean AND operator is there
Alfa + 2 * exp(Beta - Gamma - D)	complex expression

Calculating the expression you obtain a value that can be used as parameter to be assigned to a variable or a function, or as a test in a condition:

```
...
A = 3;
B = ValueGateA;
StartScale (log(14), 4 + ValueGateB);
if (GateAon && GateBon) then
  ActivateAlarm();
end
Delta = Delta * 2 + Alfa + 2 * exp(Beta - Gamma - D);
...
```

The operators used in the equations have, as it happens in mathematics, a different priority; so to make the writing of the equations more flexible and easy to understand you can use the round brackets.

```
AverageValue = (GateA + GateB + GateC) / 3;
```

Be careful not to combine logical expressions with arithmetic expressions.
Logical expressions are often where you need a test of a condition, such as *while* loops and *if..then* conditions.

Example of correct expression:

```
if ( (A && (D == 2.50)) || (B < 10) )  then
 ...
 ...
end
```

Example of NOT correct expression:

```
bool ActiveButton();
...
A = ValoreMedio * 3.14 + PulsanteAttivo();
```

The above expression is incorrect because ActiveButton() returns a logical value that can't be combined with one arithmetic.

Notes:

If an expression is compound by a Boolean operator (&& or ||) and two or more sub-expressions operating on variables of type not homogeneous among them, it is necessary to use the brackets () for each sub-expression.

Example of correct expression:

```
...
int A = 1;
int B = 5;
string S1 = "900";
string S2 = "1000";
if (( A > B )  &&  ( S1 > S2 )) then
...
end
```

Example of NOT correct expression:

```
...
int A = 1;
int B = 5;
string S1 = "900";
string S2 = "1000";
if ( A > B  &&  S1 > S2 ) then
...
end
```

10.8 Operators

The operators are language elements that link different expressions so as to allow the formation of more sophisticated ones.
There are logic operators (And, Or, …) whose result is *true* or *false*, and algebraic operators (+,-,…) whose result is a number.

OPERATOR	FUNCTION
Generic:	
()	priority
Logic	
==	equality
!=	inequality
>=	majority or equality
<=	minority or equality
>	majority
<	minority
&&	AND
||	OR
Algebraic	
+	addition
-	subtraction
*****	product
/	division

Notes:
- For a list of other available mathematical functions, refer to chapter API (program interface) - Math.
- You can use addition operator also for strings.