SCRIPTS -INDEX

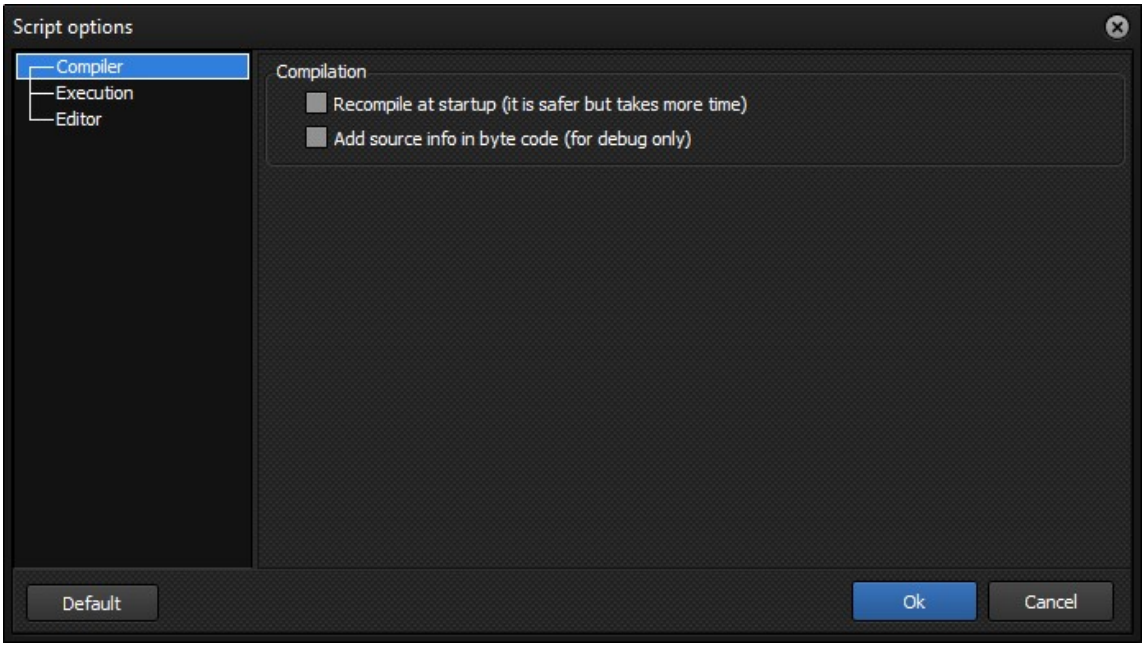
Script options

The options window is accessible from the script editor in the Tools/Options menu.

The options are divided into three parts which are:

- Compiler
- Execution
- Editor

Compiler



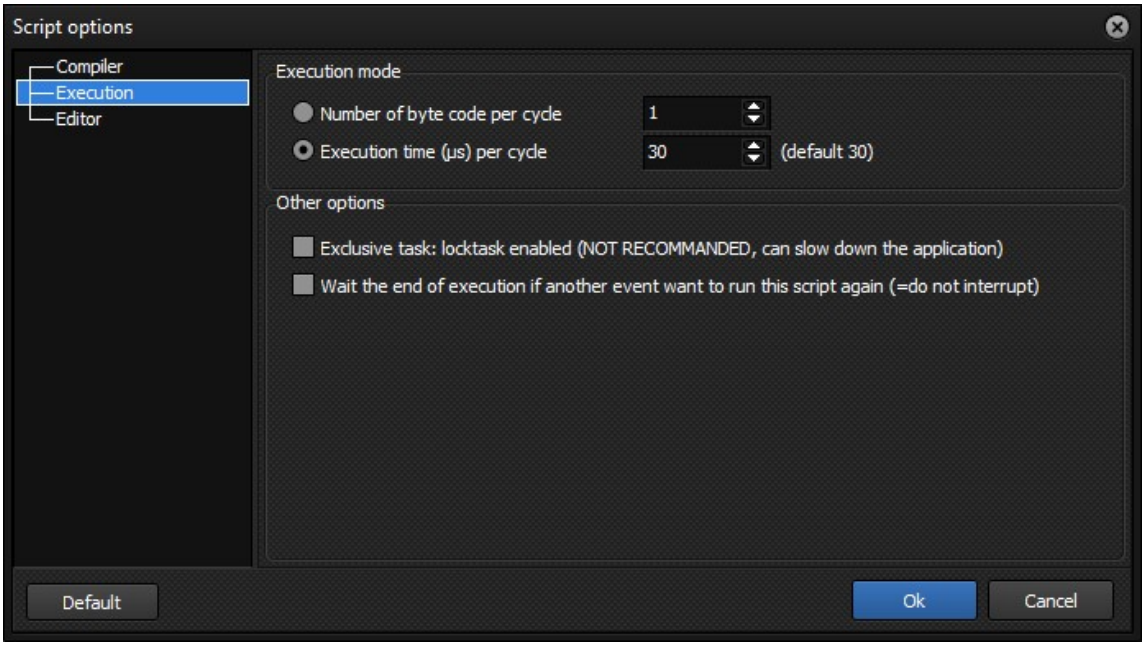
- **Recompile at startup**

When this option is checked, the script is compiled each time before being executed. This allows to check the entire script and especially the presence of all properties (tag) before the execution of the script to avoid errors at the time of execution. The disadvantage of this option is that compiling takes up CPU time and can degrade Crystal's performance. It is not recommended to activate this option if the script is executed regularly at a fast frequency.

- **Add source info in byte code**

Check this box if you want to run the script in debug mode. The debug mode allows you to run the script step by step, to visualize the content of the variables in real time in order to solve an execution problem. To execute the script in debug mode, click on the debug button located in the tool bar of the script editor.

Execution



About execution mode:

Crystal XE gives control to all scripts that are running sequentially. All scripts are executed sequentially at the compiled code level (byte code) which gives the impression that all scripts are running simultaneously in parallel. It is possible to act on the execution time that is allocated to each script with these two options. You have to be

careful when acting on these options because it can degrade the performance of other scripts and therefore of Crystal XE.

- **Number of byte code per cycle**

By choosing this option, you will be sure that a minimum number of instructions is executed in each cycle.
This option is dependent on the performance of the computer. The more powerful the computer is, the faster the script will run. If you don't want to depend on the performance of the computer, you should choose the second option.

- **Execution time**

This is the default choice. The default time allocated by cycle is 30µs and can be modified. By increasing this value you run the risk of degrading the performance of other scripts.

- **Exclusive task**

The role of this option is the same as the “Locktask” instruction but has an immediate effect on the script launch.
This option allows to concentrate all the processor power on the execution of this script. The execution of all other task are suspended. However, there is a timeout to avoid the application to freeze. To end the locking process, call the sleep() function with any value or call the unlocktask function.
As this function can have repercussions on the execution of other scripts, it is necessary to control it well because it could strongly degrade the performance of Crystal XE.

- **Wait the end of exection if another event want to run this script again**

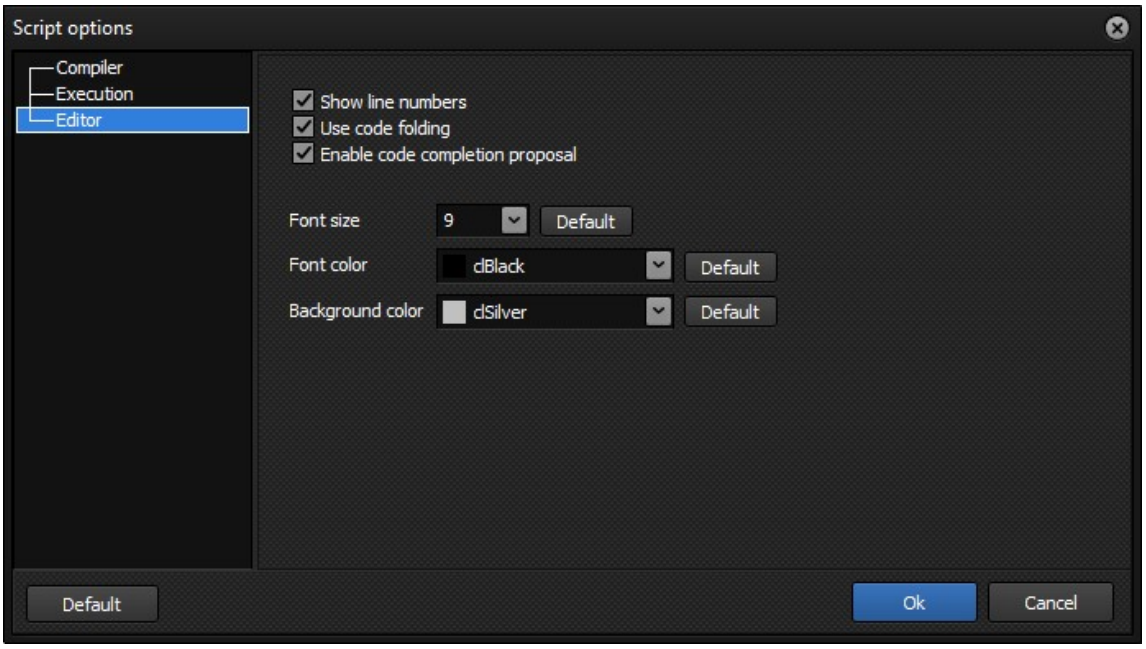
When this option is enabled, it is not possible to restart the script before the end of execution.

Example of a script running when a button is clicked

```
var i : integer;
Begin
  for i:=1 to 10 do
  Begin
    Writeconsole(intToStr(i));
    sleep(1000);
  end;
End;
```

In this example, by default, the box being unchecked, the script will be restarted from the beginning each time the button is clicked.
If the option is enabled, then nothing will happen if the user clicks the button several times while the script is already running. This prevents a script from being interrupted anywhere before the normal end of execution.

Editor options



All these options are valid for all scripts because they concern the only script editor that is accessible for all scipts.

- **Show line numbers**

Check this options to display the line numbers in the gutter (checked by default).

- **Use code folding**

Check this box if you want the [+] and [-] signs to appear in the gutter, allowing you to collapse or expand the procedure and function code or the start and end statement.

- **Enable code completion proposal**

Code Completion (Ctrl+Space) is a Code Insight feature available in the script editor, which makes easy to complete your code. Code Completion displays a resizable "hint" window that lists valid elements that you can select to add to your code. The hint window shows a list where the first symbols are the ones that start with the text you type, followed by the symbols that contain the rest of text.

Automatic code completion is enabled by default but can be disabled if, for example, it takes too long to open.
When automatic code completion is enabled, you can always uses CTRL+SPACE to invoke code completion. The code completion will also be invoke when typing a period (.)

- **Other options**

Font size, Font color and Background color are used to define the look and feel of the editor.

The Pascal language

Based on Pascal language, its benefits include easy-to-read code, quick compilation, and the use of multiple unit files for modular programming.
This language is no case sensitive, so identifiers with different cases are the same (MyVar is the same as myVAR).

Scripts contain some significant language features:
 Built in Data Types - Scripts contains it's own built in data types of Integer, Real, Character, Boolean, String.
 Supports Structured Programming - This is accomplished through the use of subprograms called procedures and functions.
 Powerful functions - Script language contains mathematical functions (log, ln, sin, cos, tan, sqrt…) data file access, logical, sound, timers, string conversion …

Compiler

Script language is a pseudo interpreted language. Before executing a program a compilation is performed to transform the script into a specific byte code. This byte code contains simple instructions which are executed by the interpreter engine.

Program organization

A script begins always by the reserved word **begin** and terminates by the other reserve word **end;**

Example

```
begin
  ShowMessage('Hello world');
end;
```

Display this message:

Before the begin instruction, it is possible to declare variables, constants, procedure or functions, and uses clause.

Comments

Comments are pieces of text used to annotate a program. Comments are for the programmers used only; they are stripped from the source text before parsing.
The compiler supports several methods; there are three ways to delineate comments.
One way is a single line comment and two ways are multiple lines comments.

Single line comment:
The compiler allows a single-line comment using two adjacent slashed (//). The comment can start in any position, and extends until the next new line.

Example of a single-line comment

```
Begin // this is a comment
   ShowMessage('Hello world'); // this will display a message
end; // this is the end of script
```

Multiple lines comments:
A multiple lines comments is any sequence of characters placed after the symbol pair (*. The comment terminates at the first occurrence of pair *) following the initial (*

Example of a multi-lines comment

```
(*
 ----------------------
 This is comment on several lines
 The program begin here
 ---------------------------
*)
Begin
  ShowMessage('Hello world');
end;
```

The following characters can also be used to delineate multiple-lines comments { and }

Other example of a multiple-lines comment

```
{
 ----------------------
 This is comment on several lines
 The program begin here
 ---------------------------
}
Begin
  ShowMessage('Hello world');
end;
```

Variables

A variable is an identifier whose value can change at runtime. Put differently a variable is a name for a location in memory; you can use the name to read or write to the memory location.

Declaring variables

The basic syntax for a variable declaration is:

```
Var identifierlist: type;
```

Where identifier is a comma-delimited list of valid identifiers and type is any valid type.

Other example of a multiple-lines comment

```
Var
  i : integer; // Declares one variable of type interger with the name "i"
  x,y : real;  // Declares two variables x and y of type real.
Begin
  // your code here
End;
```

Consecutive variable declarations do not have to repeat the reserved word var

```
Var
 x,y,z : real;  // 8 bytes float
  I,j,k : integer; // 4 bytes signed integer (32 bits)
 Ok : Boolean; // one byte Boolean
 W  : word; // unsigned word of 2 bytes (16 bits)
 S  : string; // string which contain up to 255 characters
 S1 : string[10]; // string which contain up to 10 characters
```

Assign variables:

You must use ":=" to assign a variable.

Syntax

identifier := value;

Example

```
var
  I : integer;
Begin
  I:=23;
End;
```

Array of variables:

In Crystal, arrays allow a developer to refer to a series of variables by the same name and to use a number—an index—to tell them apart.
In most scenarios, you declare an array as a variable, which allows for array elements to be changed at run-time.

In Crystal, Array of variables are limited to one dimension.

Syntax:

```
Var identifierlist: array[x..y] of type;
```

With y>x and x>=0

Example

```
var
  Buf : array[1..1000] of byte;
  I : integer;
Begin
  For i:=1 to 1000 do Buf[i]:=0;
End;
```

Record of variable (=structure)

Records are a useful and distinguishing feature of pascal. They provide a very neat way of having named data structures - groups of data fields. Unlike arrays, a record may contain different types of data.

Example

```
Var
  Temperature : record
     Name : string;
     MinTemp,MaxTemp : real;
   end;
begin
   with Temperature do
   begin
     name := 'Substrate holder';
     MinTemp := 20;
     MaxTemp := 1500;
   end;
   With Temperature do
     ShowMessage('Temperature range of '+name+' = '+IntToStr(MinTemp)+'/'+IntToStr(MaxTemp));
end;
```

Types

The following types can be used:

Type	Range	Occupied size (in bytes)
Byte	0 .. 255	1
Char	Character code 0 .. 255	1
Word	0 .. 65535	2
Boolean	0 .. 1	1

Integer or Int32	-2147483648 .. 2147483647	4
Int64	-2^63 .. 2^63-1	8
Real	Double precision floating point value. 15 significant digits. Exponent -308 to +308	8
String	Maximum length: 254 characters. Each character code : 0 .. 255	255 including the length char
String[x]	1<=x<=128 Each character code : 0 .. 255	x+1

Please note that it is not possible to create your own type, only the predefined types can be used.

Hexadecimal values

To specify a hexadecimal value in your program, insert the character $ before the value.

Example

```
Var
  i:integer;
begin
  I := $41;  // this is equivalent to 65
end;
```

Strings

A single character (char) is useful when parsing text, one character at a time. However, to handle words and sentences and screen labels and so on, strings are used. A string is literally a string of characters.

- String constants are delimited by quotes (')
- To concatenate several strings together, use the sign +.
- To insert a special character, use the sign # followed by the ASCII code number in decimal.
- To insert a special character in hexadecimal, use the sign #$ followed by the ASII code number in hexadecimal.

Example

```
Var
  Source, targer, last : string;
Begin
  Source := 'Hello World';        // Assign from a string literal
  Target := source;               // Assign from another variable
  Last   := 'Don'+#39+'t do that';// insert quote in a string
end;
```

- **For other string operations, refer to the function reference guide.**

Constants

Constants are tokens representing fixed numeric or character values.
The data type of a constant is deduced by the compiler.
Hexadecimal numbers must be preceded by the character "$". Example $10 = 16 decimal.

Declaring constants

The basic syntax for a constant declaration is:

```
Const identifier = value;
```

Where identifier is a valid identifier and value is any valid value or string.

Valid value can be

- An integer decimal (ex: -2456)
- A real number (ex: -23.56)
- A hexadecimal number – begin by the character $ (ex: $23)

- A string – must be between ' and '

Color constants are provided in the file const_colors.pas. To use it in your script, just write *uses const_colors;* at the beginging of the script.

Example 1

```
Const c1=12;
```

Example 2

```
Const
  c1=12;
  C2=$34;
  C3='My string constant';
Begin
  ShowMessage(c3);
End;
```

Array of constant

Sometimes you need to declare a constant array—a read-only array. You cannot change the value of a constant or a read-only variable. Therefore, while declaring a constant array, you must also initialize it.

Example 2

```
Var
  i : integer;
Const
  Days : array[0..6] of string = ('Sun', 'Mon', 'Tue', 'Wed','Thu', 'Fri', 'Sat');
Begin
  For i:=2 to 3 do ShowMessage(Days[i]); // will display "Tue" and "Wed"
End;
```

IF

Conditional expression

Starts a conditional expression to determine what to do next.

The "**If**" keyword is used to control the flow of code depending on the logical result of the given condition.

See the section Boolean expression for more details about conditional expression on a Boolean expression.

There are two forms of the "**If**" statement - one with an else clause, the other not.

"**If**" works as follows:
If the condition is true, then the first statement is executed.
If false, then this statement is bypassed.
If there is an else statement, it is executed instead.

A condition is true only if the result of the test is different of zero.
It is possible to perform a test on all variable types expect strings.
For example, if I is integer, you can perform a test without using an operator: if I then … in that case, the condition will be true if the variable I is different of zero.

For strings comparison, uses the special functions like CompareText.

In all cases, the Statement clause must be a contained in a begin/end block if it is longer than one statement in length.

Example

```
begin
  // Illustrate a simple if statement that executes true
  If True then ShowMessage('True!');

  // Illustrate the same, but with multiple actions
  If 1 = 1 then
```

```
  begin
    ShowMessage('We now have');
    ShowMessage('multiple lines');
  end;

  // Illustrate a simple if statement that fails
  If 1 = 2 then ShowMessage('1 = 2');

  // Illustrate an if then else statement
  If False
  then ShowMessage('False');
  else ShowMessage('True');

  // Nested if statements
  If true then
  Begin
    If false then
      ShowMessage('Inner then satisfied');
    else
      ShowMessage('Inner else satisfied');
  End else
    ShowMessage('Outer else satisfied');
end;
```

Result of this example:

```
True!
We now have
multiple lines
False
Inner else satisfied
```

Case...of

A mechanism for acting upon different values of a number.

The case keyword provides a structured equivalent to a sequence of if statements on the same variable.
The case statement is more elegant, more efficient, and easier to maintain than multiple if nestings.
The **case** statement may provide a readable alternative to deeply nested **if** conditionals.

A **case** statement has the form:

```
case selectorExpression of
   caseList1: statement1;
    ...
   caseListn: statementn;
end
```

where *selectorExpression* is any expression of an ordinal type (string types are invalid) and each *caseList* is one of the following:

- A numeral, declared constant, or other expression that the compiler can evaluate without executing your program. It must be of an ordinal type compatible with *selectorExpression*. Thus, 7, **True**, $12, clBlack can all be used as *caseLists*, but variables and most function calls cannot.
- A subrange having the form *First..Last*, where *First* and *Last* both satisfy the criterion above and *First* is less than or equal to *Last*.
- A list having the form *item1, ..., itemn*, where each *item* satisfies one of the criteria above.

Each value represented by a *caseList* must be unique in the **case** statement; subranges and lists cannot overlap.

A **case** statement cannot have a final **else** clause yet with this version.

The case statement

```
case I of
   1..5: Caption := 'Low';
   6..9: Caption := 'High';
   0, 10..99: Caption := 'Out of range';
end
```

is equivalent to the nested conditional:

```
if (I>=1) and (I<=5) then
  Caption := 'Low';
else if (I>=6) and (I<=9) then
  Caption := 'High';
  else if (I = 0) or ((I>=10) and (I<=99)) then
    Caption := 'Out of range';
```

Example

```
uses stdconst;

procedure ShowColour(colour : integer);
begin
  // Use a case statement to see the colour of the passed var
  Case colour of
       clRed : ShowMessage('The colour is Red');
     clGreen : ShowMessage('The colour is Green');
      clBlue : ShowMessage('The colour is Blue');
    clYellow : ShowMessage('The colour is Yellow');
           0 : ShowMessage('The colour zero is black');
  end;
end;
//----------------------------------------
Begin
  ShowColour(clGreen);
  ShowColour(clYellow);
end;
```

It is also possible to use the Begin...end statement in each element.

Example using begin … end

```
  Case colour of
       clRed : Begin
                  ShowMessage('The colour is Red');
                  ShowMessage('This color is used to indicate an alarm');
               End;
     clGreen : ShowMessage('The colour is Green');
      clBlue : ShowMessage('The colour is Blue');
    clYellow : ShowMessage('The colour is Yellow');
  end;
```

Several constants are possible for a single case.
In this case, each constant must be separated by a comma.

Example of several constants

```
uses stdconst;

procedure ShowColour(colour : integer);
begin
  // Use a case statement to see the colour of the passed var
  Case colour of
       clRed,clGreen,clBlue : ShowMessage('The colour is Red, green or blue');
       clYellow : ShowMessage('The colour is Yellow');
              0 : ShowMessage('The colour zero is black');
  end;
end;
//----------------------------------------
Begin
  ShowColour(clGreen);
  ShowColour(clYellow);
end;
```

Boolean expression and comparison

A boolean expression has always a binary result: true or false
True and False are constants which are equal to 1 and 0.

There are other Boolean constants, see the list bellow:

Constants identifier	Equivalent Value
True, On, Open	1
False, Off, Close	0

Operators:

Operator	Description
=	Equal
<	Lower than
>	Greater than
<>	Different
<=	Lower or equal
>=	Greater or equal
Or	See bellow
And	See bellow
Xor	See bellow
not (function)	See document function list
Mod	See bellow

OR operator

The **Or** keyword is used in two different ways:

1. To perform a logical or boolean 'or' of two logical values. If either are true, then the result is true, otherwise it is false.
2. To perform a mathematical 'or' of two integers. The result is a bitwise 'or' of the two numbers. For example: 10110001 Or 01100110 = 11110111.

XOR operator

The **Xor** keyword is used in two different ways:

1. To perform a logical or boolean 'Exclusive-or' of two logical values. If they are different, then the result is true.
2. To perform a mathematical 'Exclusive-or' of two integers. The result is a bitwise 'Exclusive-or' of the two numbers. For example: 10110001 Xor 01100110 = 11010111.

AND operator

The **And** keyword is used in two different ways:

1. To perform a logical or boolean 'and' of two logical values. If both are true, then the result is true, otherwise, the result is false.
2. To perform a mathematical 'and' of two integers. The result is a bitwise 'and' of the two numbers. For example: 10110001 And 01100110 = 00100000.

MOD operator (modulus)

The **Mod** keyword gives the remainder from dividing the **Dividend** by the **Divisor**.
The whole number result of the division is ignored.

Exemple

```
var
  int : Integer;
begin
  // Divide a primary integer by 4 - Mod returns the remainder
  int := 19 Mod 4;
  ShowMessage('19 mod 4 = '+IntToStr(int)); // will display: “19 mod 4 = 3”
end;
```

Other operator: power

The operator ^ raises the Number value to the Power value.

Example

```
Var
```

```
  Value : integer;
Begin
  Value := 4 ^ 2;
  ShowValue('4 ^ 2 = ',Value);
End;
```

Result of this example:

```
4 ^ 2 = 16
```

For

Iteration loop

Syntax:

For Variable: = Integer Expression to Integer Expression do Statement

The For keyword starts a control loop, which is executed a finite number of times.

The Variable is set to the result of the 1st Expression. If the result is less than or equal to the result of the 2nd Expression (when to is specified), then the Statement is executed. Variable is then incremented by 1 and the process is repeated until the variable value exceeds the 2nd expression value.

The Statement maybe a single line, or a set of statements with a begin/end block.
Nested loops statements are allowed.

Example 1

```
Var
  i:integer;
  Tab : array[1..12] of integer;
Begin
  For i:=1 to 12 do Tab[i]:=I;
end;
```

Example 2 using nested loops

```
Var
  I,j : integer;
Begin
  For i:=1 to 2 do
  Begin
    For j:=4 to 6 do
    Begin
      showMessage('i='+intToStr(i)+' j='+intToStr(j));
    End;
  End;
end;
```

Result of example 2:

```
Iteration#1: i=1 j=4
Iteration#2: i=1 j=5
Iteration#3: i=1 j=6
Iteration#4: i=2 j=4
Iteration#5: i=2 j=5
Iteration#6: i=2 j=6
```

Repeat

Syntax:
Repeat
  Statement1;
{statement2;
…}
Until Expression;

The Repeat keyword starts a control loop that is always executed at least once, and which terminates when the Expression is satisfied (returns True).
There is no need for Begin or End markers - the Repeat and Until keywords serve that purpose.
It is used when it is important that the statements are at least executed once.

Example

```
Var
  R :real;
Begin
  Repeat
    R := QueryValue('Enter a value (0 to exit)');
    If R<>0 then ShowMessage(REalToStr(1/R,10));
  Until R=0;
end;
```

Goto

Forces a jump to a label, regardless of nesting.

The Goto keyword forces a jump to a given label.
It should Never be used code since it makes code very difficult to maintain.
It is mostly used to force a termination of heavily nested code, where the logic to safely exit would be tortuous.

Example

```
Var
  i : Integer;
begin
  for i := 1 to 10 do
  begin
    ShowMessage('i = '+IntToStr(i));
    if i = 4 then Goto Label1;   // Conditionally exit the loop
  end;
  ShowMessage('The loop finished OK');
Label1:
  ShowMessage('Loop finished with i = '+IntToStr(i));
end;
```

With command

The **With** keyword is a convenience provided by Crystal Pascal language for referencing elements of a complex variable, such as a record or property (tags).
It simplifies the code by removing the need to prefix each referenced element with the complex variable name.

For example :
myRecord.colour := clRed;
myRecord.size   := 23.5;
myRecord.name   := 'Fred';

can be rewritten :

With myRecord **do**
begin
  colour := clRed;
  size   := 23.5;
  name   := 'Fred';
end;

Several elements can be written and must be separated by commas, for example :

With myRecord1,MyRecord2,MyRecord3,App.Var,Growth.As_P2 **do**
Begin
(…)
End;

However be warned that it can surprisingly make your code more difficult to read, especially when nesting with clauses. More disturbingly, it can create maintenance problems, where a code change can mean that the wrong target for the 'child' field referenced.

Procedures and functions

A subroutine is like a sub-program. It not only helps divide your code up into sensible, manageable chunks, but it also allows these chunks to be used (called) by different parts of your program. Each subroutine contains one of more statements.

In common with other languages, scripts provide 2 types of subroutine - Procedures and Functions. Functions are the same as procedures except that they return a value in addition to executing statements. A Function, as its name suggests, is like a little program that calculates something, returning the value to the caller. On the other hand, a procedure is like a little routine that performs something, and then just finishes.

Parameters to subroutines

Both functions and procedures can be defined to operate without any data being passed. For example, you might have a function that simply returns a random number. It needs no data to get it going.

Likewise, you can have a procedure that carries out some task without the need for data to dictate its operations.

Often, however, you will pass data, called parameters, to a subroutine. (Note that the definition of a subroutine refers to parameters as arguments - they are parameters when passed to the subroutine).

Some simple function and procedure examples

Example of procedure declaration that not included parameter

```
Procedure showTime;
Begin
  ShowMessage('the current date and time is'+GetDateTime(''));
End;

Begin // let us call this procedure
  ShowTime; // will display in a window: 26/02/2013 17:15:16
End;
```

Example of procedure declaration with parameters

```
Procedure showTime;
Begin
  ShowMessage('the current date and time is'+GetDateTime(''));
End;

Begin // let us call this procedure
  ShowTime; // will display in a window: 26/02/2013 17:15:16
End;
```

Example of function declaration without parameters

```
function RandomChar:char;
var I : integer;
Begin
  I := random(255);
  Result := chr(i);
End;

Begin // let us call this procedure
  ShowMessage(Randomchar);
End;
```

Uses

The Uses keyword defines a list of one or more files that are used by the current script.
Each files are effectively imported.
This keyword can be located anywhere in the program but outside of a procedure, a function or a statement (for, repeat, if ..) however, it is advisable to write this keyword at the beginning of the program.
It is not necessary to add the extension for each file.
If the extension is missing, the default extension is .pas

If the path is not specified then the search directories are (in order of priority):

- Project recipe directory (<Project dir>\recipe\)
- If the script is a recipe, the search path is the chamber recipe directory (<Project dir>\recipe\<Chamber name>)
- If the script belongs to a form, the search path is the path where the form is located.
- The project script directory (<Project dir>\Script\)
- The project templates script directory (<Project dir>\Template\Script\)
- The program templates script directory (<Program dir>\Template\Script\)
- The project templates report directory (<Project dir>\Template\Report\)
- The program templates report directory (<Program dir>\Template\Report\)

Crystal XE is provided with included files that you can use in your script:
- Dateutils.pas: provide date and time utilities functions
- StdConst.pas: collection of standard constants
- DDE_report: DDE library to create Word or Excel reports
- OLEConst.pas: Collection of OLE constants. See section about OLE for more details.

Example to use the file MyLibrary.pas

```
Uses MyLibrary;

Begin
  MyFunction1;  // Myfunction1 is declared in the file Mylibrary.pas
  MyFunction2;  // Myfunction2 is declared in the file Mylibrary.pas
End;
```

Included files

Crystal XE is provided with source files which located in the template directory that you can uses in your scripts.

Constant colors defines in the StdConst.pas file

About 20 predefined color constants are provided in this file:

Colour constant	Meaning	Hexadecimal value	Example
clBlack	Black	$000000	
clMaroon	Maroon	$000080	
clGreen	Green	$008000	
clOlive	Olive Green	$008080	
clNavy	Navy Blue	$800000	
clPurple	Purple	$800080	
clTeal	Teal	$808000	
clGray	Grey	$808080	
clSilver	Silver	$C0C0C0	
clRed	Red	$0000FF	
clLime	Lime Green	$00FF00	
clYellow	Yellow	$00FFFF	
clBlue	Blue	$FF0000	
clFuchsia	Fuchsia	$FF00FF	
clAqua	Aqua	$FFFF00	
clLtGray	Light Grey	$C0C0C0	clSilver alias
clDkGray	Dark Grey	$808080	clGray alias
clWhite	White	$FFFFFF	
clCream	Cream	$F0FBFF	
clMedGray	Medium Grey	$A4A0A0);	
clMoneyGreen	Mint Green	$C0DCC0	
clSkyBlue	Sky Blue	$F0CAA6	

- For more information, see the files located in the directory <program directory>Template/Script

Functions reference

Index

APP = Application methods

Methods

- SaveSecurities
- SetPlatenNumber

For PLC functions like platen position, see also the **Batch** methods.

ChamberPortToId

Returns the order number (id) of the chamber connected to a PLC port.

Format

Function **ChamberPortToId**(PortNumber: integer):integer;

Parameters

- PortNumber: Number of the port of the cluster or the pick and place beginning at 1.

Returned value

The order number of the chamber or -1 if the port is not linked to a chamber.

Description

The function **ChamberPortToId** returns the chamber order number of the chamber identified by the port number.

Example

```
Var
  idx : integer;
Begin
  idx := ChamberPortToId(3);
  ShowMessage(GetChamberNameId(idx));  // Display the name of the chamber connected to the port 3 of t
he cluster
End;
```

See also: Application methods

ClearAllPlatens

Clear the platens of all chambers.

Format

Procedure **ClearAllPlatens**;

Description

The function **ClearAllPlatens** reset all platens in all chamber (set all position to zero).

Example

```
Begin
   App.clearAllPlatens;
End;
```

In this example, all platens of the project are reset.

See also: Application methods

ClearPlatens

Clear platens position in the chamber identified by the chamber number.

Format

Procedure **ClearPlatens**(ChamberNumber: integer);

Parameters

- ChamberNumber: Number of the chamber beginning at **zero**.

Description

The function **ClearPlatens** reset all platens in a chamber (set all position to zero).

Example

```
Var
  i : integer;
Begin
  for i:=0 to app.chambercount-1 do
  Begin
    App.clearPlatens(i);
  end;
End;
```

In this example, all platens of the project are reset.

See also: Application methods

GetChamberNameId

Returns the name of the chamber identified by the index passed in parameter.

Format

Function **GetChamberNameId**(Idx: integer):String;

Parameters

- Idx: Index of the chamber (= chamber order number), beginning at zero.

Returned value

The name of the chamber identified by Idx.

Description

The function **GetChamberNameId** returns the name of a chamber identified by the chamber order number Idx.

Example

```
Var
  i : integer;
Begin
  for i:=0 to app.ChamberCount-1 do
  Begin
    WriteConsole(App.GetchamberNameId(i));
  end;
End;
```

See also: Application methods

GetPlatenNumber

Returns the number of a platen everywhere in the system.

Format

Function **GetPlatenNumber**(ChId, Position: integer):Integer;

Parameters

- ChId: Index of the chamber (= chamber order number), beginning at **zero**.
- Position : position of the platen in the chamber (cassette) beginning at **one**.

Returned value

The number of the platen or zero.

Description

The function **GetPlatenNumber** returns the number of a platen identified by the chamber number and the position in that chamber.

This example displays the platen number in the first chamber at the first position.

Example

```
Begin
  WriteConsole(IntToStr(App.GetPlatenNumber(0,1)));
End;
```

See also: Application methods

LoadSecurities

Loads the security configuration from a file

Format

Function **LoadSecurities**(Filename:string):boolean; // default directory is the project directory.

Parameters

- FileName: Security file name to read

Returned value

Returns true if successful

Description

The function **LoadSecurities** loads the configuration of all securities from an xml file.
The Default directory is the project directory but a full path can be defined.
The Default extension is .sec

Example

```
Begin
  App.LoadSecurities('John_securities');
End;
```

See also: Application methods

SaveSecurities

Saves the security configuration in a file

Format

Function **SaveSecurities**(Filename:string):boolean; // default directory is the project directory.

Parameters

- FileName: Output file name

Returned value

Return true if successful

Description

The function **SaveSecurities** saves the configuration of all securities in an xml file.
The Default directory is the project directory but a full path can be defined.
The Default extension is .sec

Example

```
Begin
  App.SaveSecurities('John_securities');
End;
```

See also: Application methods

SetPlatenNumber

Define the number of a platen.

Format

Procedure **SetPlatenNumber**(ChId, Position, number: integer);

Parameters

- ChId: Index of the chamber (= chamber order number), beginning at **zero**.
- Position : position of the platen in the chamber (cassette) beginning at **one**.
- Number: number of the platen to define.

Description

The function S**etPlatenNumber** defines the number of a platen identified by the chamber number and the position in that chamber.

This example define the platen number 3 in the first chamber at the first position.

Example

```
Begin
  App.SetPlatenNumber(0,1,3);
End;
```

See also: Application methods

ASCII server

The following methods are only available in the script of the ASCII server.
The ascii server can be accessed from the options window in the Servers item, then ASCII servers.

Methods

- XML_Answer : send a XML frame to the last connected client.
- XML_ReceiveBuf : Assign the received string into an XML document.
- XML_Send : Sends an XML document to a server.
- STR_Answer : Sends a string frame to the last connected client.

STR_Answer

(USED BY ASCII SERVER ONLY)

Sends an string frame to the client

Format

Function STR_Answer(Frame :string):Boolean;

Parameters

- **Frame**: String character (max 250 characters).

Returned value

The function returns true if successfully completed.

Description

Use the function STR_Answer (in the event OnReceiveStr) after receiving a request from the host computer which is connected to the ASCII Server to sent a response to the request.

Example of script in the ASCII server

```
// This script is defined in the event OnReceiveStr(ServerId:integer; RxFrameStr:string)
Begin
  writeconsole(StrToASCI(RxFrameStr));
  STR_Answer('Ok'+#13);
End;
```

See also: ASCII server

ASCII client module

The following methods are only available in the script of the ASCII driver.

Methods

- EnablePort: Enable, Disable the serial or socket module connected to the ASCII module
- GetTagFieldStrEx: Returns the content of the column of a tag
- GetTagID: Returns the identifier of the tag specified by its name
- ReadBuffer: Read the contents of the receive buffer and possibly clear it
- ReadTagFromDevice : Read a tag in the device
- ReadTagStr: Returns the string content of the tag specified by its identifier
- ReadTagStrEx: Returns the string content of the tag specified by its name
- ReadTagValue: Returns the value of the tag specified by its identifier
- ReadTagValueEx: Returns the value of the tag specified by its name
- RxReadFrame: Returns the last frame received for a specifed tag
- SendString: Send a frame to the device
- SendStringFormat: Send a frame to the device
- SetTagValueEx: Assigns the value to the tag specified by its name but not execute the OnWrite event and not send the value to the device.
- WriteTagStr: Assigns the string content to the tag specified by its identifier
- WriteTagStrEx: Assigns the string content to the tag specified by its name
- WriteTagValue: Assigns the value to the tag specified by its identifier
- WriteTagValueEx: Assigns the value to the tag specified by its name

EnablePort

(ASCII module)

Enable or disable the serial or socket module connected to the ASCII module.

Format

Function EnablePort(Status:int32;ShowLogMsg:boolean):int32

Parameters

- Status: Use to enable (1) or Close (0) or Destroy (-2) or return the status of the Port (-1)
- ShowLogMsg: true to log events messages when connecting/disconnecting

Returned value

- Status of the port:
  -1= No socket or serial module connected to the Modbus module.

0 = Port (socket or serial module) is Closed or destroyed
1 = Port (socket or serial module) is enabled

Description

This function is used to enable or disable (close or destroy) the port connected to the modbus module. It can also be used to know the status of the port.

- If the port is a serial port, disabling will close the com port and enabling will open it.
- If the port is a socket, closing will disconnect the socket from the server and enabling will reconnect the socket to the server. The destruction will destroy the memory socket (the socket will be re-created when it is activated)
- If no port is connected to the ASCII module then the function returns -1

Example

```
Begin
  EnablePort(0,false); // This will close or disconnect the port silently.
End;
```

See also: ASCII methods

GetTagFieldStrEx

(ASCII module)

Returns the content of a column of a tag

Format

Function GetTagFieldStrEx(TagId:word;ColName:string):string

Parameters

- IdTag: Tag identifier
- ColName: Name of the column

Returned value

- Content of the column

Description

Reads then content of a column.
The tag identifier (***IdTag***) is passed as a parameter to the script when the script is called by the event OnRead, OnWrite or OnRequest of the ASCII module.

The column name can be:

- Name, RW, Msg, Link, Frequency, fxa, fxb, min, max, default, opt

Example

```
Begin
  ShowMessage('Content of factor a of the tag TSP is '+GetTagFieldStrEx(IdTag,'fxa'));
End;
```

See also: ASCII methods

GetTagId

(ASCII module)

Returns the identifier of the tag specified by its name

Format

Function GetTagId(TagName:string;Channel:word):integer

Parameters

- TagName: String

- Channel: Channel number (starting at 1)

Returned value

The index of the tag (starts at 0)

Description

Returns the index of the tag identified by its name **TagName** and its **channel** number.
TagName is not case sensitive.
The **channel** number may be passed in parameter to the script.

Example

```
Var
  id: integer;
  R : real;
Begin
  id := GetTagId('MV',1); // MV is located at address 100
  R  := ReadTagValue(id); // Get the value of MV in the modbus table
  ShowValue('Data =', R); // Display the value of MV
End;
```

See also: ASCII methods

ReadBuffer

(ASCII module)

Read the contents of the receive buffer and optionally clear it.

Format

Function ReadBuffer(Clear:boolean):String;

Parameters

- Clear: true to clear the receive buffer.

Returned value

- Returns the contents of the receive buffer into a string

Description

This function returns the content of the receive buffer. If the parameter Clear is true, then the contents is clear just after the reading. If the frame analyser is opened then the string will be output to the analyser.
The returned string can contain printable and not printable characters.

Example

```
Begin
  ShowMessage(ReadBuffer(false)); // Display the contents of the receive buffer
End;
```

See also: ASCII methods

ReadTagFromDevice

(ASCII module)

Reads a tag in the device

Format

Function **ReadTagFromDevice**(TagName:string; Channel:word; ReadingMode:byte):integer;

Parameters

TagName: Name of the tag to send

Channel: Channel number (start at 1)
ReadingMode:
0: A reading request is initiated but the script does not wait to receive the answer, the tag will be updated later, after receiving the answer.
1: A reading request is initiated, and the script waits until receiving the answer.
2: The script does not trig a reading.

Returned value

- The tag index or -1 if the tag does not exist.

Description

This function reads a tag identified by **TagName** (not case sensitive) and **channel** (start at 1) in the device, wait or not for reading and update the table.

Returns the index of the tag or -1 if the tag does not exist (this may happen also if the channel is not connected to a sub equipment)
If the tag does not exist, an error message will be printed to the console.

Example

```
Begin
  ReadTagFromDevice('SP',1,1);
End;
```

See also: ASCII methods

ReadTagStr

(ASCII module)

Returns the String content of the tag specified by its identifier

Format

Function ReadTagStr(IdTag:word):string;

Parameters

- IdTag: Tag identifier (starting at 0)

Returned value

- String content of the tag idenfied by <STR>

Description

This function returns the content of the received string identified by <STR> in the received format.
The tag identifier is passed as a parameter to the script (TagId) when the script is called by the event OnRead, OnWrite or OnRequest of the ASCII module.

Example

```
Var
  Id : integer;
Begin
  Id := GetTagId('MV',1);
  ShowMessage(ReadTagStr(IdTag)); // Display the content of the tag 'MV'
End;
```

See also: ASCII methods

ReadTagStrEx

(ASCII module)

Returns the String content of the tag specified by its name

Format

Function ReadTagStrEx(TagName:String;Channel:word):string;

Parameters

- TagName: Name of the tag (string)
- Channel: Channel of the tag (starting at 1)

Returned value

- String content of the tag identified by <STR>.

Description

Returns the received string identified by <STR> in the received format of the Tag specified by TagName and the Channel number.
The channel number is passed as a parameter to the script when the script is called by the event OnRead, OnWrite or OnRequest of the ASCII module.

Example

```
Begin
// The channel is passed as a parameter to the script when the script is called by the event
OnRead, OnWrite or OnRequest of the ASCII module.
  ShowMessage('MV content='+ReadTagStrEx('MV',channel)); // Display the last received frame o
f the tag 'MV'.
End;
```

See also: ASCII methods

ReadTagValue

(ASCII module)

Returns the value of the tag specified by its identifier

Format

Function ReadTagvalue(IdTag:word):real;

Parameters

- IdTag: Tag identifier (starting at 0)

Returned value

- Value of the tag in a real number

Description

This function returns the value of the tag specified by IdTag.
The tag identifier is passed as a parameter to the script when the script is called by the event OnRead, OnWrite or OnRequest of the ASCII module.

Example

```
Begin
  ShowMessage( RxReadFrame(IdTag) ); // Display the last received frame of the tag 'MV'.
End;
```

See also: ASCII methods

ReadTagValueEx

(ASCII module)

Returns the value of the tag specified by its name

Format

Function ReadTagValueEx(TagName:String;Channel:word):real;

Parameters

- TagName: Name of the tag (string)
- Channel: Channel of the tag (starting at 1)

Returned value

- Value of the tag in a real number.

Description

Returns the value of the tag specified by TagName and the Channel number.
The channel number is passed as a parameter to the script when the script is called by the event OnRead, OnWrite or OnRequest of the ASCII module.

Example

```
Begin
// The channel is passed as a parameter to the script when the script is called by the event
OnRead, OnWrite or OnRequest of the ASCII module.
  ShowValue('MV content=',ReadTagValueEx('MV',channel)); // Display the last received frame
of the tag 'MV'.
End;
```

See also: ASCII methods

RxReadFrame

(ASCII module)

Returns the last frame received for a specified tag

Format

Function RxReadFrame(IdTag:word):String;

Parameters

- IdTag: Tag identifier (starting at 0)

Returned value

- Returns the last frame received

Description

This function returns the last frame received of a tag.
The tag identifier is passed as a parameter to the script when the script is called by the event OnRead, OnWrite or OnRequest of the ASCII module.
The returned string can contain printable and not printable characters depending of the protocol.

Example

```
Var
 Id : integer;
Begin
  ShowMessage( RxReadFrame(0) ); // Display the last received frame of the first tag.
  Id := GetTagId('MV',1);
  ShowMessage( RxReadFrame(Id) ); // Display the last received frame of the tag 'MV'.
End;
```

See also: ASCII methods

SendString
(ASCII module)

Sends a frame to the device

Format

Function SendString(Message:String;IsWriteRequest,WaitAnswer:boolean):string;

Parameters

- Message: Frame to send
- IsWriteRequest: true when the frame write a data to the device, false when it is to request a data.
- WaitAnswer: true to wait that the device returns a message.

Returned value

- When the parameter WaitAnswer is true, the function returns the message sent by the device.

Description

Sends a frame to the device.
The frame must be specified into the parameter ***Message***.
The Reading enquiry format as defined in the general tab is not used. Only the message specified by the parameter ***Message*** is sent to the device.

- When the parameter ***WaitAnswer*** is true, the parameter IsWriteRequest indicates if the ASCII module must wait for a reading message answer or for a writing message answer. The format for the reading or writing is defined in the General tab and in the field *Answer format* of the corresponing tab.
- When the parameter ***WaitAnswer*** is false, then the parameter ***IsWriteRequest*** is not used.

Example

```
Var
  S : String;
Begin
  S := SendString('GetData1?',false,true);
  ShowMessage('The device has returned '+S);
End;
```

See also: ASCII methods

SendStringFormat
(ASCII module)

Sends a frame to the device specifying the received frame format.

Format

Function SendStringFormat(Message,RxFormat:String; WaitAnswer:boolean):string;

Parameters

- Message: Frame to send
- RxFormat: Format of the receiving frame
- WaitAnswer: true to wait that the device returns a message.

Returned value

- When the parameter WaitAnswer is true, the function returns the message sent by the device.

Description

Sends a frame to the device.

The frame must be specified into the parameter ***Message***.
The Reading enquiry format as defined in the general tab is not used. Only the message specified by the parameter ***Message*** is sent to the device.
If the RxFormat is Empty then the Reading enquiry format will be used but, in that case it is recommended to use the function SendString.
The received frame must be conformed of the RxFrame parameter.

Example

```
Var
  S : String;
Begin
  S := SendStringFromat('G?'+#13,'<STR><CR>',true); // The answer terminates by CR.
  ShowMessage('The device has returned '+S);
End;
```

See also: ASCII methods

SendTagValueEx

(ASCII module)

Assign the value of the tag with the value parameter

Format

Function SendTagValueEx(TagName:String; Channel:word; Value:Real):string;

Parameters

- TagName: Name of the tag
- Channel: Channel of the tag
- Value: Value to assigne.

Description

Example

```
Var
  S : String;
Begin
  S := SendStringFromat('G?'+#13,'<STR><CR>',true); // The answer terminates by CR.
  ShowMessage('The device has returned '+S);
End;
```

See also: ASCII methods

SetTagValueEx

(ASCII module)

Assigns the value of the tag specified by its name but not run the OnWrite script.

Format

Procedure SetTagvalueEx(TagName:String;Channel:word;Value:Real);

Parameters

- TagName: Name of the tag
- Channel: Channel number (starting at 1)
- Value: Value to assign to the tag

Description

This function is similar to the WriteTagValueEx but not execute the OnWrite script and not send the value to the device.

Example

```
Begin
  SetTagValueEx('TSP',1,10);
End;
```

See also: ASCII methods

WriteTagStr

(ASCII module)

Assigns the string content of the tag specified by its identifier

Format

Procedure WriteTagStr(IdTag:word;Str:Srting;SendToDevice:Byte);

Parameters

- IdTag: Tag identifier (Starting at zero)
- Str: String to assign to the tag
- SendToDevice: 0 or false =don't send it to the device, 1 or true=write the tag to the device, 2=write the tag to the device and wait for the value to be sent.

Description

This function can be used to change the string content of the tag specified by IdTag.
The tag identifier is passed as a parameter to the script when the script is called by the event OnRead, OnWrite or OnRequest of the ASCII module.

Example

```
Begin
  WriteTagStr(TagId,'Result is 12.45 degree',true);
End;
```

See also: ASCII methods

WriteTagStrEx

(ASCII module)

Assigns the string content of the tag specified by its name

Format

Procedure WriteTagStrEx(TagName:String;Channel:word;Str:String;SendToDevice:Byte);

Parameters

- TagName: Name of the tag
- Channel: Channel number (starting at 1)
- Str: String to assign to the tag
- SendToDevice: 0 or false =don't send it to the device, 1 or true=write the tag to the device, 2=write the tag to the device and wait for the value to be sent.

Description

This function is similar to the WriteTagStr but the tag is identified by its name.
The channel number is passed as a parameter to the script when the script is called by the event OnRead, OnWrite or OnRequest of the ASCII module.

Example

```
Begin
  WriteTagStr('TSP',1,'This is the tag content',true);
End;
```

See also: ASCII methods

WriteTagValue

(ASCII module)

Assigns the value of the tag specified by its identifier

Format

Procedure WriteTagValue(IdTag:word;Value:real;SendToDevice:Byte);

Parameters

- IdTag: Tag identifier (Starting at zero)
- Value : Value to assign to the tag
- SendToDevice: 0 or false =don't send it to the device, 1 or true=write the tag to the device, 2=write the tag to the device and wait for the value to be sent.

Description

This function can be used to change the value of the tag specified by IdTag.
The tag identifier is passed as a parameter to the script when the script is called by the event OnRead, OnWrite or OnRequest of the ASCII module.
The function will apply the relation (***Value-FxB)/FxA*** only when sending the value to the device.

Example

```
Begin
  WriteTagValue(TagId,100,true);
End;
```

See also: ASCII methods

WriteTagValueEx

(ASCII module)

Assigns the value of the tag specified by its name

Format

Procedure WriteTagvalueEx(TagName:String;Channel:word;Value:Real;SendToDevice:Byte);

Parameters

- TagName: Name of the tag
- Channel: Channel number (starting at 1)
- Value: Value to assign to the tag
- SendToDevice: 0 or false =don't send it to the device, 1 or true=write the tag to the device, 2=write the tag to the device and wait for the value to be sent.

Description

This function is similar to the WriteTagValue but the tag is identified by its name.
The channel number is passed as a parameter to the script when the script is called by the event OnRead, OnWrite or OnRequest of the ASCII module.
The function will apply the relation (***Value-FxB)/FxA*** only when sending the value to the device.

Example

```
Begin
  WriteTagValue('TSP',1,10,true);
End;
```

See also: ASCII methods

Batch (Automation: App.Batch also available for all clusters)

The App.Batch object can be used when there is only one transfer automaton (Called also PLC for Programmable Logic Controller) on the system. If, on the other hand, there are several transfer PLC, the cluster object functions must be used.
For example, if the system contains two clusters named Cluster1 and Cluster2 respectively, then all the methods starting with App.Batch can be used for the Cluster1 and Cluster2 objects. So, to start the Cluster2 batch, simply write Cluster2.Start.

Methods

- ClearAlarmTable
- GetAlarmTable
- GetPlatenName
- GetPlatenPosition
- LoadFromFile
- Log
- Pause
- Resume
- RunRecipe
- SetAlarmTable
- Start
- Stop

ClearAlarmTable

Clear the PLC alarm table

Invocations

- App.Batch.ClearAlarmTable
- <ClusterName>.ClearAlarmTable

Format

Procedure <object>.**ClearAlarmTable**(ClearStatusAlarm:boolean);

Parameter

- ClearStatusAlarm: true to clear also the flag StatusAlarm

Description

Remark: The relative device (cluster or pick and place) must be connected to a server (not a socket), if not, the writing is not allowed.
This procedure clear all the alarm table in the modbus table of the device (cluster or pick and place)
If the parameter “ClearStatusAlarm” is enabled then the tag StatusAlarm is also reset and the PLC alarm disappears.

Example

```
Begin
  App.batch.ClearAlarmTable(true);
End;
```

See also: Batch

GetAlarmTable

Read an alarm in the PLC

Invocations

- App.Batch.GetAlarmTable
- <ClusterName>.GetAlarmTable

Format

Function <object>.**GetAlarmTable**(AlarmNumber: word):byte;

Parameter

- Alarm number (1 to 512 for a cluster or 1 to 438 for a pick and place)

Description

This procedure reads an alarm in the modbus alarm table of the relative device (cluster or pick and place).

For a cluster the relative modbus table is the following:

Holding registers 351 to 606	351 : MSB=Alarm #1 & LSB= Alarm #257 352 : MSB=Alarm #2 & LSB= Alarm #258 … 605 : MSB=Alarm #255 & LSB= Alarm #511 606 : MSB=Alarm #256 & LSB= Alarm #512

For a pick and place, the relative modbus table is the following:

Holding registers 32 to 249	32: MSB=Alarm #1 & LSB= Alarm #220 33: MSB=Alarm #2 & LSB= Alarm #221 … 248: MSB=Alarm #218 & LSB= Alarm #437 249: MSB=Alarm #219 & LSB= Alarm #438

In this example, we set the bit #1 (bitmap value 2) of the alarm 12:

Example

```
Var
  b : byte;
Begin
  b := App.Batch.GetAlarmTable(12);
  b := b OR 2;
  App.Batch.SetAlarmTable(12,b,true);
End;
```

See also: Batch

GetPlatenName

Returns the name (PlatenId) of a platen in the current batch

Invocations

- App.Batch.GetPlatenName
- <ClusterName>.GetPlatenName

Format

Function **App.Batch.GetPlatenName**(PlatenNumber:Integer):String;

Parameter

- Platen number: number of the platen

Returned value

- PlatenId

Description

This function returns the platenId (string of characters entered by the user) or an empty string if the PlatenNumber is not found.

Example

```
Begin
  ShowValue(Platen identifiant number 1 =',App.Batch.GetPlatenName(1));
End;
```

See also: Batch

GetPlatenPosition

Returns the position of a platen in the current batch

Invocations

- App.Batch.GetPlatenPosition
- <ClusterName>.GetPlatenPosition

Format

Function **App.Batch.GetPlatenPosition**(PlatenNumber:Integer):Integer;

Parameter

- Platen number: number of the platen

Returned value

- The chamber number (x1000) + the position or -1

Description

This function returns the position of the selected platen following this relation:
The chamber number (x1000) + the position.
- Ex 3005 is the chamber 3 (means the port of the cluster) and the position 5.

It returns -1 if the platen was not found or is in an intermediate position.

Example

```
Begin
  ShowValue('Position of platen 1 =',App.Batch.GetPlatenPosition(1));
End;
```

See also: Batch

SetAlarmTable

Store an alarm in the PLC

Invocations

- App.Batch.SetAlarmTable
- <ClusterName>.SetAlarmTable

Format

Procedure <object>**.SetAlarmTable**(AlarmNumber: word; AlarmValue:byte;SetStatusAlarm:boolean);

Parameter

- Alarm number (1 to 512 for a cluster or 1 to 438 for a pick and place)
- AlarmValue: bitmap of the alarm (8 bits) to set.
- SetStatusAlarm: true to set the StatusAlarm flag (=alarm number for a cluster or 256 for a pick and place)

Description

Remark: The relative device (cluster or pick and place) must be connected to a server (not a socket), if not, the writing is not allowed.

This procedure stores an alarm in the modbus alarm table of the relative device (cluster or pick and place).

For a cluster the relative modbus table is the following:

Holding registers 351 to 606	351 : MSB=Alarm #1 & LSB= Alarm #257 352 : MSB=Alarm #2 & LSB= Alarm #258 … 605 : MSB=Alarm #255 & LSB= Alarm #511 606 : MSB=Alarm #256 & LSB= Alarm #512

For a pick and place, the relative modbus table is the following:

Holding registers 32 to 249	32: MSB=Alarm #1 & LSB= Alarm #220 33: MSB=Alarm #2 & LSB= Alarm #221 … 248: MSB=Alarm #218 & LSB= Alarm #437 249: MSB=Alarm #219 & LSB= Alarm #438

Example

```
Begin
  App.batch.SetAlarmTable(1,5,true); // Set bit 0 and 2 of the alarm#1
End;
```

See also: Batch

Start

Starts a batch

Invocations

- App.Batch.Start
- <ClusterName>.Start

Format

Function <object>**.Start** : Boolean;

Returned value

The function returns true is success and false is no batch is loaded or if an error occurred.

Description

Starts the current batch.
To load a batch, call the function **LoadFromFile** before.

Example

```
Var
  Ok : boolean;
  i : integer;
Begin
  Ok := App.Batch.LoadFromFile('test');
  if Ok then Ok := App.Batch.Start;;
  if Ok then ShowMessage('Success');
        else ShowMessage('Error starting the batch "test" or file not found');
End;
```

See also: Batch

Stop

Stops a batch

Invocations
- App.Batch.Stop
- <ClusterName>.Stop

Format

Procedure <object>**.Stop**;

Description

Stops the current execution of batch without any condition.
This function has the same effect as pushing the Stop button in the batch inspector.

Example
```
Begin
  App.Batch.Stop;
End;
```

See also: Batch

Pause

Pauses a batch

Invocations
- App.Batch.Pause
- <ClusterName>.Pause

Format

Procedure <object>**.Pause**;

Description

Pauses the current execution of batch without any condition.
This function has the same effect as pushing the Pause button in the batch inspector.

Example
```
Begin
  App.Batch.Pause;
End;
```

See also: Batch

Resume

Resumes a batch

Invocations
- App.Batch.Resume
- <ClusterName>.Resume

Format

Procedure <object>**.Resume**;

Description

If the batch is paused, this procedure Resume the current execution of batch.
This function has the same effect as pushing the Resume button in the batch inspector.

Example
```
Begin
  App.Batch.Resume;
End;
```

See also: Batch

Log (batch)

Logs a line in the batch log

Invocations

- App.Batch.Log
- <ClusterName>.Log

Format

Procedure <object>**.Log**(Horodate:Boolean; Message: String);

Description

Stores a text line in the history list of the batch.
This function can be executed when the batch is running or not.

Example

```
Begin
  App.Batch.log(true,'The gate is opened');
End;
```

See also: Batch

LoadFromFile (batch)

Loads a batch file

Invocations

- App.Batch.LoadFromFile
- <ClusterName>.LoadFromFile

Format

Function <object>**.LoadFromFile**(Filename:string):boolean;

Parameter

- FileName: String;

Returned value

- True if the file is loaded (the file is found and the current batch is stopped)

Description

Loads a batch file.
The current batch must be stopped.

Example

```
Var
  Ok : boolean;
  i : integer;
Begin
  Ok := App.Batch.LoadFromFile('test');
  if Ok then Ok := App.Batch.Start;;
  if Ok then ShowMessage('Success');
        else ShowMessage('Error starting the batch "test" or file not found');
End;
```

See also: Batch

MovePlaten (batch)

Move a platen from a position to another position

Invocations

- App.Batch.MovePlaten
- <ClusterName>.MovePlaten

Format

Function <object>.**MovePlaten**(FromChamber:string;FromPos:integer;ToChamber:string;ToPos:integer; WaitEnd:boolean):integer;

Parameter

- FromChamber: Name of departure chamber
- FromPos: Departure position (position number 1 to n)
- ToChamber: Name of destination chamber
- ToPos: Destination position (position number 1 to n)
- WaitEnd : true if the script must wait for the end of the move.

Returned value

- Not used yet;

Description

Move a platen from a position to another.
Both start and finish positions must be accessible from the same transfer controller (e.g. in multi-cluster applications).
If a recipe is running in the start position, the function will wait for the recipe to finish before moving the platen.

Conditions

- The current batch scheduler must be running.
- A platen must be present in the start position.
- The destination position must be free.

→ If a condition is not met, an error occurs and the batch stops.

Example

```
Begin
  App.Batch.MovePlaten('Load1',3,'Growth',1,true);
  // or
  Cluster1.MovePlaten('Load1',1,'MBE1',1,true);
  // or if the script is executed in the batch inspector of cluster 1
  MovePlaten('Load1',1,'MBE1',1,true);
  // or
  Cluster2.MovePlaten('Load2',2,'MBE2',1,true);
  ShowMessage('The platen is now in the growth chamber');
End;
```

See also: Batch

RunRecipe (batch)

Execute a recipe in a chamber

Invocations

- App.Batch.RunRecipe
- <ClusterName>.RunRecipe

Format

Function <object>.**RunRecipe**(Chamber,FileName : string; BitmapOptions:word; WaitEnd:boolean) : integer

Parameter

- **Chamber**: Name of chamber
- **Filename**: Recipe file name to load
- **BitmapOptions**: see bellow
- **WaitEnd**: true if the script must wait for the end of the recipe.

Returned value

- Not used yet;

Description

Bit organization of BitmapOptions:
- b0 = execute pre recipe
- b1 = execute post recipe

Values of BitmapOptions
- 0 = no other recipe is executed.
- 1 = the pre-recipe is executed.
- 2 = the post-recipe is executed.
- 3 = the pre-recipe and the post-recipe are executed.

The RunRecipe function loads the recipe identified by Filename into the recipe inspector and launches it. Depending on bitmapOptions, pre-recipe and/or post-recipe will be executed. The names of the pre- recipe and post-recipe are defined in the CrystalXE options.

Example

```
Begin
  App.Batch.RunRecipe('Growth','RecipeTestLayers.rcp',0,true);
  ShowMessage('The recipe is now terminated');
End;
```

See also: Batch

Camera

For more details about image processing, see the dedicated manual which can be downloaded on the Internet site (http://www.crystalxe.com)

Methods

- iBrightness
- iGetImage
- iGrayScale
- iInvert
- iCompare
- iContrast
- iGamma
- iGetRheedVLinesCount
- iProjection
- iRelease
- iScanLineEdge
- iSmooth
- iThreshold
- LoadFromFile
- SaveToFile
- Start
- Stop

SaveToFile

(Camera method)

Saves a snapshot to an image file

Format

Procedure SaveToFile(FileName:string;WithTimeStamp:boolean);

Parameters

FileName: Name of the file to create. The default extension is .JPG but you can also specify .BMP
WithTimeStamp: True if the name of the file begins by the date and time

Description

This procedure saves a snapshot into an image file.

Known file types:
- Jpeg: **.JPG (default)**
- Bitmap: **.BMP**
- Portable Network Graphics: **.PNG**

Example
```
// Save an image every seconds in JPG format
Begin
  Camera1.Start;
  ResetTimer(1);
  repeat
    if GetTimer(1)>1 then
    Begin
      resetTimer(1);
      Camera1.SaveToFile('TimeLapse',true);
    end;
  until (tag=0);
End;
```

See also: Camera methods

Stop
(Camera method)

Stops the live camera

Format

Procedure Stop;

Description

Stops the display of the camera (or webcam).

Example
```
// Save an image every seconds in JPG format
Begin
  Camera1.Start;
  ResetTimer(1);
  repeat
    if GetTimer(1)>1 then
    Begin
      resetTimer(1);
      Camera1.SaveToFile('TimeLapse',true);
    end;
  until (tag=0);
  Camera1.stop;
End;
```

See also: Camera methods

Start
(Camera method)

Starts the camera

Format

Procedure Start;

Description

This function starts the display of the live camera.

Example

```
// Save an image every seconds in JPG format
Begin
  Camera1.Start;
  ResetTimer(1);
  repeat
    if GetTimer(1)>1 then
    Begin
      resetTimer(1);
      Camera1.SaveToFile('TimeLapse',true);
    end;
  until (tag=0);
  Camera1.stop;
End;
```

See also: Camera methods

LoadFromFile

(Camera method)

Loads and image file

Format

Function LoadFromFile(FileName:string):boolean;

Parameters

FileName: Name of the file to open with the complete path. The extension of the file name must be provided.

Returned value

Returns true if success, otherwise the function returns false.

Description

Loads an image file into the input buffer.
The image is not displayed.
To display the image, use the function iGetimage.

Known file types:

- **JPG:** jpeg (compressed with loss)
- **BMP**: bitmap (lossless)
- **GIF**: Graphic interchange format (lossless)
- **PNG**: Portable Network Graphics (compression lossless - recommended)
- **TIF**: Tagged Image file (lossless)

Example

```
Begin
  Camera1.LoadFromFile('c:\images\picture1.png');
  Camera1.iGetimage(true);
End;
```

See also: Camera methods

iGetImage

(Camera method)

Loads an image into the display buffer.

Format

Procedure iGetImage(refresh:boolean);

Parameters

Refresh: if true then the display will be refreshed

Description

Copies an image from the input buffer to the display buffer.
Set the parameter “refresh” to false only if you want to execute other processing functions before to display it. This will increase the performance.

Example

```
Begin
  Camera1.LoadFromFile('c:\images\picture1.png');
  Camera1.iGetimage(true);
End;
```

See also: Camera methods

iGrayScale

(Camera method)

Applies the gray scale filter to the display buffer.

Format

Procedure iGrayScale;

Description

Transforms the display buffer into a gray scale image.

Example

```
Begin
  Camera1.LoadFromFile('c:\images\picture1.png');
  Camera1.iGetimage(true);
  Camera1.iGrayScale;
End;
```

See also: Camera methods

iThreshold

(Camera method)

Transforms the display buffer into a two bits image (black and white)

Format

Procedure iThreshold (Threshold:integer);

Parameters

Threshold: value between 0 to 255

Description

Transforms each pixel in gray scale and apply the threshold to transform the entire image into a 2 bits image.

All pixel value lower than the threshold become black and all others become white.

Example
```
Begin
  Camera1.LoadFromFile('c:\images\picture1.png');
  Camera1.iGetimage(true);
End;
```

See also: Camera methods

iInvert
(Camera method)

Inverts the image pixel by pixel

Format

Procedure iInvert;

Description

Inverts each pixel with the function: New pixel value = 255 – current value;

Example
```
Begin
  Camera1.LoadFromFile('c:\images\picture1.png');
  Camera1.iGetimage(true);
  Camera1.iInvert;
End;
```

See also: Camera methods

iContrast
(Camera method)

Change the contrast of the display buffer.

Format

Procedure iContrast(contrast:integer);

Parameters

contrast: value between 0 and 65535

Description

Apply a contrast filter to the display buffer.

Example
```
Begin
  Camera1.LoadFromFile('c:\images\picture1.png');
  Camera1.iGetimage(true);
  Camera1.iContrast(2000);
End;
```

See also: Camera methods

iBrightness
(Camera method)

Change the brightness of the display buffer.

Format

Procedure iBrightness(brightness:integer);

Parameters

brightness: value between -3000 to +3000

Description

Apply a brightness filter to the display buffer

Example

```
Begin
  Camera1.LoadFromFile('c:\images\picture1.png');
  Camera1.iGetimage(true);
  Camera1.iBrightness(2000);
End;
```

See also: Camera methods

iGamma

(Camera method)

Applies a gamma filter to the display buffer.

Format

Procedure iGamma(value: real);

Parameters

value: gamma value

Description

Applies a gamma filter to the display buffer.

Example

```
Begin
  Camera1.LoadFromFile('c:\images\picture1.png');
  Camera1.iGetimage(true);
  Camera1.iGamma(0.6);
End;
```

See also: Camera methods

iCompare

(Camera method)

Compares two images.

Format

Function iCompare(ImageFileName:String; Mode,Options : int32):real;

Parameters

- ImageFileName: Image file name to compare with the current image in the display buffer.
- Mode: Comparison mode, see description below.
- Options: option depending of the mode.

Returned value

Returns the result of the comparison.

Description

Compares two image files and return the result of the comparison.

- If mode = 0 (CM_histo) then the comparison will use the histogram method applied on the gray scale images. The result will be the sum of the difference for each pixel value (0 to 255).

The lower the returned number, the closer the image will be.

Example

```
Var
  Comp : real;
Begin
  Camera1.LoadFromFile('c:\images\picture1.png');
  Camera1.iGetimage(true);
  Comp := Camera1.iCompare('c:\images\picture2.png',0,0);
  ShowValue('Result of comparison is: ',Comp);
End;
```

See also: Camera methods

iRelease

(Camera method)

Ends image processing.

Format

Procedure iRelease;

Description

After the call of this procedure, the display buffer will be always the mirror of the input buffer.

Example

```
Begin
  Camera1.iRelease;
End;
```

See also: Camera methods

iScanLineEdge

(Camera method)

Highlights vertical lines using scan line method.

Format

Procedure iScanLineEdge(Level,HMin:int32);

Parameters

- Level: Threshold applied to each scan line to detect a change in intensity (Example: 6)
- VMin: Eliminate all vertical lines lower than Vmin (Example: 10)
- HMin: Eliminate all horizontal lines lower than Hmin (Example: 4)

Description

For each line of the gray scale image, from left to right, if the intensity of a pixel increases more than the ***level*** parameter, then next pixels will be set to 255 (white). If the intensity decreases, then next pixels will be black. So image resulting will be in two bits (black and white)

A filter is applied to eliminate vertical lines that are less wide than VMin.
Another filter is applied to eliminate horizontal lines that are less wide than HMin.

Example

```
Begin
  Camera1.LoadFromFile('c:\images\picture1.png');
  Camera1.iGetimage(true);
  // Add 3 rotation buttons BtnLevel, BtnVMin and BtnHMin to adjust the parameters
  Camera1.iScanlineEdge(BtnLevel.Position,BtnVMin.Position,BtnHMin.Position);
// default parameters can be Camera1.iScanlineEdge(6,10,4)
End;
```

See also: Camera methods

iProjection

(Camera method)

Computes a projection of the image on X or Y axis.

Format

Function iProjection(Axis, threshold, width: word) : int32;

Parameters

- Axis: 0 to project on the X axis, 1 to project on the Y axis
- Threshold: Threshold applied on the average value
- Width: with of the graphical representation of the projection

Returned value

The function returns the number of edges.
The average distance between all edges can be found in the property Camera.iparam1

Description

The Axis parameter indicates the type of projection.

If Axis=0, the projection will be applied on the first line of the X axis (top of the image). In that case, for each column, the average pixel is calculated and the Threshold parameter is applied to this value. If the average value is greater than the threshold then the pixel will be white, otherwise it will be black. To be visible, the first line is copied followed the parameter Width.

If Axis=1, the projection will be applied on the first column of the Y axis (left of the image). In that case, for each line, the average pixel is calculated and the Threshold parameter is applied to this value. If the average value is greater than the threshold then the pixel will be white, otherwise it will be black. To be visible, the first column is copied followed the parameter Width.

Example

```
Var
  n : integer;
Begin
  Camera1.LoadFromFile('c:\images\picture1.png');
  Camera1.iGetimage(true);
  Camera1.iScanlineEdge(6,10,4);

  n := Camera1.iProjection(0,50,20);
  ShowValue('Number of edges=',n);
  ShowValue('Gap average=',Camera1.iparam1);

End;
```

See also: Camera methods

iSmooth

(Camera method)

Apply a smooth filter on the image.

Format

Procedure iSmooth(mode,intensity:integer);

Parameters

- Mode: 0 or 1, see description
- intensity: The higher the intensity, the greater the effect.

Description

The moving average algorithm filter is applied on each line of the gray scale image.

If mode = 0:
For each pixel of each line the following formula is applied:
Pixel intensity = (previous pixel intensity + (Pixel intensity)*Intensity + Next pixel intensity) / (2/Intensity);

If mode = 1
For each pixel of each line the following formula is applied:
Pixel intensity = (previous pixel intensity + Next pixel intensity) / 2;

Example

```
Begin
  Camera1.LoadFromFile('c:\images\picture1.png');
  Camera1.iGetimage(true);
  Camera1.iSmooth(0,2);
End;
```

See also: Camera methods

iGetRheedVLinesCount

(Camera method)

Returns the number of vertical lines found in the image for Rheed purpose.

Format

Function GetRheedVLinesCount(MinPeak,ThresholdLevel,LinesCount,VMin,HMin:int32):int32;

Parameters

- MinPeak:
- Level: This parameter is used to detect vertical lines in the image, see function iScanLineEdge for more details.
- **LinesCount**: Number of lines used to calculate the average number of peak detection.
- **VMin**: Minimum vertical line height.
- **HMin**: Minimum horizontal line width

Returned value

Number of vertical lines found in the image.

Description

The function searches the first row in which the number of peak is higher than the MinPeak parameter and this from the top of the image to the bottom direction.
Then the function does the same but from the bottom of the image to the top direction.

Between these two lines, the function will calculate the average of peak found in the **Linescount** lines.

Example

```
Var
  NbPeak : integer;
Begin
  Camera1.LoadFromFile('c:\images\picture1.png');
  Camera1.iGetimage(true);
```

```
  NbPeak := Camera1.GetRheedVLinesCount(2,5,50,2,4);
  if NbPeak<0
    then ShowMessage('Error '+IntToStr(NbPeak));
    else ShowMessage('Number of lines='+IntToStr(NbPeak));
End;
```

See also: Camera methods

Chambers

Methods

- ClearAllPlatens
- GetCellsTemperature
- GetEquipmentName
- GetPlatenNumber
- GetSubEquipmentName
- SetPlatenNumber

ClearAllPlatens

Clear all platens of the given chamber.

Format

Procedure ClearAllPlatens;

Description

This function clear all platens of the given chamber.
This function must be preceded by the name of the relative chamber.

Example

```
Begin
  Growth.ClearAllPlatens;
End;
```

See also: **Chambers methods**

GetCellTemperature

Get the temperature of cells => TO BE COMPLETED

Format

Function **GetCellsTemperature(bUsedCellOnly : boolean; Separator:string) : string;**

Parameter

- **bUsedCellOnly**: when true, the shutter must be opened, when false, the shutter must not exists for this cell.

Returned value

- The name of the equipment followed by the temperature values separated by the separator.

Description

This function returns the name of the equipment and the temperature of the cells.

Example

```
Begin
  WriteConsole(Growth.GetCellTemperature(true,';'));
```

```
End;
```

See also: Chambers methods

GetEquipmentName

Get the name of equipment identified by the index

Format

Function **GetEquipmentName(idx:integer) : string;**

Parameter

- Idx: the number of equipment beginning at zero.

Returned value

- The name of the equipment identified by the index passed in parameter.

Description

This function returns the name of the equipment identified by the index passed in parameter.
This function must be preceded by the name of the relative chamber.

Example

```
Var
  i : integer;
Begin
  for i:=0 to Growth.EquipmentCount-1 do
  Begin
    WriteConsole(Growth.GetEquipmentName(i));
  end;
End;
```

See also: Chambers methods

GetPlatenNumber

Get the number of the platen identified by its position

Format

Function **GetPlatenNumber(position:integer) : integer;**

Parameter

- Position: The position of the platen from 1.

Returned value

- The platen number.

Description

This function returns the number of the platen in the chamber identified by the position passed in parameter.
This function must be preceded by the name of the relative chamber.

Example

```
Var
  i : integer;
Begin
  WriteConsole('Number of position in loading='+IntToStr(Loading. NumberOfPosition));
  for i:=1 to Loading.NumberOfPosition do
```

```
  Begin
    WriteConsole(IntToStr(Loading.GetPlatenNumber(i)));
  end;
End;
```

See also: Chambers methods

GetSubEquipmentName

Get the name of equipment identified by the index

Format

Function **GetSubEquipmentName(EquipmentIndex, SubEquipmentIdx:integer) : string;**

Parameter

- EquipmentIndex: the number of equipment beginning at zero.
- SubEquipmentIndex: the number of sub equipment beginning at zero

Returned value

- The name of the sub equipment identified by the index passed in parameter.

Description

This function returns the name of the sub equipment identified by the index of the equipment and the index of the sub equipment passed in parameter.
This function must be preceded by the name of the relative chamber.

Example

```
Var
  i,j,SubCount : integer;
  TagName : String;
Begin
  for i:=0 to C21DZ.EquipmentCount-1 do
  Begin
    TagName := 'C21DZ.'+C21DZ.GetEquipmentName(i)+'.SubEquipmentCount';
    SubCount:=GetPropValue(TagName,0);
    for j:=0 to SubCount-1 do
    Begin
      WriteConsole('Equ#'+IntToStr(i+1)+'/ Sub#'+IntToStr(j)+'='+C21DZ.GetSubEquipmentName(i,
j));
    end;
  end;
End;
```

See also: Chambers methods

SetPlatenNumber

Define the platen number identified by its position

Format

Procedure **SetPlatenNumber(position,Number:integer);**

Parameter

- Position: The position of the platen from 1.
- Number: The Number of the platen.

Description

This function defines the number of the platen in the chamber identified by the position passed in parameter.
This function must be preceded by the name of the relative chamber.

Example

```
Begin
```

```
  Loading1.SetPlatenNumber(1,12);// The first platen is number 12
End;
```

See also: **Chambers methods**

Charts

Methods

- AddCurve
- AddPoint
- ClearAll
- ClearCurve
- CopyToClipBoard
- CurveColor
- DeleteCurve
- DeletePoint
- Detach
- GetCount
- GetX
- GetY
- InsertLabel
- LoadFromFile
- SaveToFile
- SetScaleLimit
- SetX
- SetY
- ShowCurve

AddCurve

(Chart method)

Add a new curve in a chart

Format

Procedure AddCurve(CurveName,EquX,EquY,Trig:String; Style:Int32);

Parameters

- CurveName: Name displayed in the legend
- EquX: Equation for X Axis (Relevant only if the chart is not based on a timescale)
- EquY: Equatio for Y axis
- Trig: Trigger.

Please note that these parameters (EquX, EquY and Trig) are only useful for automatic acquisition. You can leave strings empty in case the curve points are added by a script.

Description

Add a new curve in a chart. The color of the curve is assigned automatically but can be changed by the procedure CurveColor.

Example

```
Begin
  Chart1.AddCurve(
          'Ga Temperature', // Label
          '', // Nothing in X axis
          'Growth.Ga_P3.Tip_Temperature.MV', // Equation for Y axis
          'Laps(1000)', // Trigger
        0)// Style
End;
```

See also: Chart functions

AddPoint

(Chart method)

Adds a new point in a curve

Format

Procedure AddPoint(CurveNumber: Byte; X,Y: real ; Style: byte);

Parameters

CurveNumber: Curve number beginning at 1.
X: Value of X coordinate of the new point
Y: Value of Y coordinate of the new point
Style: 0=linked to the preview point, 1=break

Description

Adds a new point in the spécified curve with coordinates “X” and “Y” and the style “Style”
If the X axis is relative to the clock time, then the X coordinate is not used, the current time is used.

Style can be:
0 = The new point is linked to the previous point by drawing a solid line.
1 = The point is not linked to the previous point, the curve is broken at this point.
4 = The point is linked to the previous point with a dotted line.

Example

```
Begin
  Chart1.AddPoint(1,3,4,0);
End;
```

See also: Chart functions

ClearAll

(Chart method)

Clears all curves in the chart

Format

Procedure ClearAll;

Description

Clears all curves of the chart.

Example

```
Begin
  Chart1.ClearAll;
End;
```

See also: Chart functions

ClearCurve

(Chart method)

Clears a curve in a chart

Format

Procedure ClearCurve(CurveNumber:byte);

Parameters

CurveNumber: Curve number beginning at one.

Description

Clears the specified curve.

Example
```
Begin
  Chart1.clearCurve(1);
End;
```

See also: Chart functions

CopyToClipBoard
(Chart method)

Copies the chart bitmap to the clipboard

Format

Procedure CopyToClipBoard;

Description

Copy the chart into the clipboard in bitmap format.

Example
```
Begin
  Chart1.CopyToClipBoard;
End;
```

See also: Chart functions

CurveColor
(Chart method)

Changes the current color of a curve

Format

Procedure CurveColor(CurveNumber : Byte;Color : int32);

Parameters

CurveNumber: Curve number beginning at 1.
Color: RGB color

Description

Changes the current color of the specified curve number. All the point added after this instruction will be with this color.
Color format = BBGGRR with BB is hexadecimal byte for blue, GG is hexadecimal byte for green, RR is hexadecimal byte for red.
Color example:
Red = $FF
Green = $FF00
Blue = $FF0000
Cyan = $FFFF00
White = $FFFFFF
Black = 0

Example
```
Begin
  Chart1.CurveColor(1,$FF); // Change to red
  Chart1.AddPoint(1,3,200,0);
End;
```

See also: Chart functions

DeleteCurve
(Chart method)

Delete a curve in a chart

Format

Function DeleteCurve(CurveNumber:byte):int32;

Parameters

CurveNumber: Curve number beginning at one or zero to request the number of curves.

Returns

Returns the number of curves or -1 if the parameter passed is out of range.

Description

Delete the specified curve.

Example
```
Begin
  ShowMessage('Nbr of curves after deletion is '+IntToStr(Chart1.DeleteCurve(1));
End;
```

See also: Chart functions

DeletePoint
(Chart method)

Deletes a point in a curve

Format

Procedure DeletePoint(CurveNumber: Byte; PointNumber: integer);

Parameters

CurveNumber: Curve number beginning at 1.
PointNumber: PointNumber to delete begenning at 1.

Description

Deletes a point of the specified curve.

Example
```
Begin
  Chart1.DeletePoint(1,4);
End;
```

See also: Chart functions

Detach
(Chart method)

Detach a chart to a floating form

Format

Function Detach(Title:String;mode,x,y,w,h:integer):boolean;

Parameters

Title : Title of the window

Mode : The mode can take several values
0: Position defined by the parameters x and y in the current monitor.
1: Top and left are the mouse cursor position.
2: Window centered on the mouse cursor position.
3: Maximized
x,y : Position of the window (only useful when mode=0)
w,h : width and height of the window

Returned value

Returns true if the chart is detached and false if the chart is returns to its original position.

Description

Detach the chart in a resizable floating form.
If the form was already detached before calling this function or if the form is closed by pressing the cross button, then the form returns to its original position.

Example

```
Begin
  Chart1.Detach('My chart',1,0,0,400,400);
End;
```

See also: Chart functions

GetCount

(Chart method)

Returns the number of point of a curve

Format

Function Getcount(CurveNumber: byte): word;

Parameters

CurveNumber: Curve number beginning at 1.

Returned value

This function returns the number of points.

Description

This function returns the number of point of the specified curve number.

Example

```
Begin
  ShowValue('Total point number of the first curve is ',Chart1.Getcount(1));
End;
```

See also: Chart functions

GetX

(Chart method)

Reads X coordinate of a point in a curve

Format

Function GetX(CurveNumber:byte; PointNumber:word):Real;

Parameters

CurveNumber: Curve number beginning at 1.
PointNumber: Number of point beginning at 1.

Description

Reads X coordinate of a point in a curve.
The curve is specified by CurveNumber and the point by PointNumber.

Example

```
Var
  X : real;
Begin
  X := Chart1.GetX(1,1);
  ShowValue('X=',X);
End;
```

See also: Chart functions

GetY

(Chart method)

Reads Y coordinate of a point in a curve

Format

Function GetY(CurveNumber:byte; PointNumber:word):Real;

Parameters

CurveNumber: Curve number beginning at 1.
PointNumber: Number of point beginning at 1.

Description

Reads Y coordinate of a point in a curve.
The curve is specified by CurveNumber and the point by PointNumber.

Example

```
Var
  Y : real;
Begin
  Y := Chart1.GetY(1,1);
  ShowValue('Y=',Y);
End;
```

See also: Chart functions

InsertLabel

(Chart method)

Inserts a label

Format

Procedure InsertLabel(CurveNumber:byte; PointNumber:word;Label:string);

Parameters

CurveNumber: Curve number beginning at 1.
PointNumber: Point number beginning at 1.
Label: String of characters.

Description

Inserts a label in the specified curve and point.

Example

```
Begin
  Chart1.InsertLabel(1,1,'Start of curve');
End;
```

See also: Chart functions

loadFromFile

(Chart method)

Loads a chart from a CSV file

Format

Function LoadFromFile(FileName:string):Boolean;

Parameters

FileName: Name of the file to open. The file must be a CSV file created by the function SaveToFile or by the operator directly by using the interface of the object.

Returned value

- True if succeed.
- False if a file operation is already in progress.

Description

Loads a chart from a CSV file.
Current curves will be cleared and replace by the new ones.

Example

```
Begin
  If not(Chart1.LoadFromFile('test.csv')) then ShowMessage('Error reading file or file format
in not recognized');
End;
```

See also: Chart functions

SaveToFile

(Chart method)

Saves a chart to a CSV file

Format

Function SaveToFile(FileName:string):Boolean;

Parameters

FileName: Name of the file to create. The file format will be a CSV file like the one created by the operator directly by using the interface of the object.

Returned value

- True if succeed.
- False if a file operation is already in progress.

Description

Saves a chart into a CSV file.
All curves of the chart will be saved.

Example

```
Begin
  If not(Chart1.SaveToFile('test.csv')) then ShowMessage('Error writing file');
End;
```

See also: Chart functions

SetScaleLimit
(Chart method)

Makes a zoom limit

Format

Procedure SetScaleLimit;

Description

This function changes the scale of the chart to make a zoom limit and display all the points of all the curves.

Example
```
Begin
  Chart1.SetScaleLimit;
End;
```

See also: Chart functions

SetX
(Chart method)

Changes X coordinate of a point in a curve

Format

Procedure SetX(CurveNumber : byte;PointNumber:Word;X:Real);

Parameters

CurveNumber: Curve number beginning at 1.
PointNumber: Number of point beginning at 1.
X : Value of X coordinate to assign

Description

Changes X coordinate of a point in a curve.
The curve is specified by CurveNumber and the point by PointNumber.

Example
```
Begin
  Chart1.SetX(1,1,34);
End;
```

See also: Chart functions

SetY
(Chart method)

Changes Y coordinate of a point in a curve

Format

Procedure SetY(CurveNumber : byte;PointNumber:Word;Y:Real);

Parameters

CurveNumber: Curve number beginning at 1.
PointNumber: Number of point beginning at 1.
Y : Value of Y coordinate to assign

Description

This procedure changes Y coordinate of a point in a curve.
The curve is specified by CurveNumber and the point by PointNumber.

Example

```
Begin
  Chart1.SetY(1,1,34);
End;
```

See also: Chart functions

ShowCurve

(Chart method)

Shows or hide a curve

Format

Procedure ShowCurve(CurveNumber:byte; Show:boolean);

Parameters

CurveNumber: Curve number beginning at 1.
Show = true to show the curve and false to hide the curve.

Description

Shows or hide the curve specified by CurveNumber.

Example

```
Begin
  Chart1.ShowCurve(1,true);
End;
```

See also: Chart functions

Date / time

Methods

- FormatDateTime
- GetDateTime
- GetNow
- MilliSecondsBetween
- StrToDateTime
- WaitDateTime

GetDateTime

Gets the current date and/or time

Format

Function **GetDateTime**(Formating:string):string;

Parameters

- Formating: Date/Time format to return

Returned value

- The current date and/or time (ex: 'dd/mm/yy tt') in string format

Description

The GetDateTime Function provides rich formatting of the current date/time into a string Result. Formatting is defined by the **Formating** string.

The **Formating** string can comprise a mix of ordinary characters (that are passed unchanged to the result string), and data formatting characters. This formatting is best explained by the example code.

The following (non-Asian) formatting character strings can be used in the Formatting string:

y = Year last 2 digits
yy = Year last 2 digits
yyyy = Year as 4 digits
m = Month number no-leading 0
mm = Month number as 2 digits
mmm = Month using ShortDayNames (Jan)
mmmm = Month using LongDayNames (January)
d = Day number no-leading 0
dd = Day number as 2 digits
ddd = Day using ShortDayNames (Sun)
dddd = Day using LongDayNames (Sunday)
ddddd = Day in ShortDateFormat
dddddd = Day in LongDateFormat

c = Use ShortDateFormat + LongTimeFormat
h = Hour number no-leading 0
hh = Hour number as 2 digits
n = Minute number no-leading 0
nn = Minute number as 2 digits
s = Second number no-leading 0
ss = Second number as 2 digits
z = Milli-sec number no-leading 0s
zzz = Milli-sec number as 3 digits
t = Use ShortTimeFormat
tt = Use LongTimeFormat

am/pm = Use after h : gives 12 hours + am/pm
a/p = Use after h : gives 12 hours + a/p
ampm = As a/p but TimeAMString,TimePMString

Example with date

```
Var
  formattedDateTime : string;
begin
  // Date only - numeric values with no leading zeroes (except year)
  formattedDateTime := GetDateTime('d/m/y');
  ShowMessage('              d/m/y = '+formattedDateTime);

  // Date only - numeric values with leading zeroes
  formattedDateTime := GetDateTime('dd/mm/yy');
  ShowMessage('           dd/mm/yy = '+formattedDateTime);

  // Use short names for the day, month, and add freeform text ('of')
  formattedDateTime := GetDateTime('ddd d of mmm yyyy');
  ShowMessage('  ddd d of mmm yyyy = '+formattedDateTime);

  // Use long names for the day and month
  formattedDateTime := GetDateTime('dddd d of mmmm yyyy');
  ShowMessage('dddd d of mmmm yyyy = '+formattedDateTime);

  // Use the ShortDateFormat settings only
  formattedDateTime := GetDateTime('ddddd');
  ShowMessage('              ddddd = '+formattedDateTime);

  // Use the LongDateFormat settings only
  formattedDateTime := GetDateTime('dddddd');
  ShowMessage('             dddddd = '+formattedDateTime);

  // Use the ShortDateFormat + LongTimeFormat settings
  formattedDateTime := GetDateTime('c');
  ShowMessage('                  c = '+formattedDateTime);
end;
```

Example with time

```
var
  formattedDateTime : string;
begin
  // Time only - numeric values with no leading zeroes
  formattedDateTime := GetDateTime('h:n:s.z');
```

```
  ShowMessage('     h:n:s.z = '+formattedDateTime);

  // Time only - numeric values with leading zeroes
  formattedDateTime := GetDateTime('hh:nn:ss.zzz');
  ShowMessage('hh:nn:ss.zzz = '+formattedDateTime);

  // Use the ShortTimeFormat settings only
  formattedDateTime := GetDateTime('t');
  ShowMessage('           t = '+formattedDateTime);

  // Use the LongTimeFormat settings only
  formattedDateTime := GetDateTime('tt');
  ShowMessage('          tt = '+formattedDateTime);

  // Use the ShortDateFormat + LongTimeFormat settings
  formattedDateTime := GetDateTime('c');
  ShowMessage('           c = '+formattedDateTime);
end;
```

See also: DateTime functions

WaitDateTime

Waits for the next date and/or time

Format

Procedure **WaitDateTime**(DateTime:string);

Parameters

- DateTime: Date/Time to wait for.

Description

The procedure WaitDateTime stops the script until the **DateTime** is reached.
The format for Datetime can be
- based on the format **"dd/mm/yy hh:mm:ss"**
- or based on the format **"dd/mm/yyyy hh:mm:ss"**

Example of **DateTime**:
- 31/12/15 02:25
- 31/12/2015 02:12:12
- 12:25: will wait until the next 12:25 (same day or next day)

Example

```
Begin
  WaitDateTime('12:25');
  ShowMessage('time is reached');
End;
```

See also: DateTime functions

StrToDateTime

Converts a date time string into a real

Format

Function **StrToDateTime**(DateTime,Format:string):Real;

Parameters

- DateTime: Date/Time in string
- Format: define the Date/Time format of the parameter DateTime

Description

The function StrToDateTime returns a real value of the Data/time given into parameter.

Example

```
Var
  R,R1,R2 : Real;
Begin
  R1 := StrToDateTime('25/02/2018 15:25:23,456','dd/mm/yyyy hh:mm:ss:zzz');
  R2 := GetNow;
  R := MilliSecondsBetween(R1,R2);
  ShowMessage(IntToStr(R));
End;
```

See also: DateTime functions

FormatDateTime

Converts a date time string into a real

Format

Function **FormatDateTime**(Format:String; DataTime:Real):string;

Parameters

- Format : define the Date/Time format of the parameter DateTime
- DateTime: Date/Time in real

Returns

- The date and time in string

Description

The formatdatetime function provides rich formatting of a DateTime value into a string. Formatting is defined by the Format string.

The Formatting string can comprise a mix of ordinary characters (that are passed unchanged to the result string), and data formatting characters. This formatting is best explained by the example code.

The following formatting character strings can be used in the Format string:

y = Year last 2 digits
yy = Year last 2 digits
yyyy = Year as 4 digits
m = Month number no-leading 0
mm = Month number as 2 digits
mmm = Month using ShortDayNames (Jan)
mmmm = Month using LongDayNames (January)
d = Day number no-leading 0
dd = Day number as 2 digits
ddd = Day using ShortDayNames (Sun)
dddd = Day using LongDayNames (Sunday)
ddddd = Day in ShortDateFormat
dddddd = Day in LongDateFormat

c = Use ShortDateFormat + LongTimeFormat
h = Hour number no-leading 0
hh = Hour number as 2 digits
n = Minute number no-leading 0
nn = Minute number as 2 digits
s = Second number no-leading 0
ss = Second number as 2 digits
z = Milli-sec number no-leading 0s
zzz = Milli-sec number as 3 digits
t = Use ShortTimeFormat
tt = Use LongTimeFormat

am/pm = Use after h: gives 12 hours + am/pm
a/p = Use after h: gives 12 hours + a/p

ampm = As a/p but TimeAMString, TimePMString
/ = Substituted by DateSeparator value
: = Substituted by TimeSeparator value

Example

```
Var
  R,R1,R2 : real;
  S : string;
Begin
  S := GetDateTime('dd/mm/yyyy hh:mm:ss,zzz');
  R1 := StrToDateTime(S,'dd/mm/yyyy hh:mm:ss:zzz');
  sleep(100);
  S := GetDateTime('dd/mm/yyyy hh:mm:ss,zzz');
  R2 := StrToDateTime(S,'dd/mm/yyyy hh:mm:ss:zzz');
  R := MilliSecondsBetween(R1,R2);
  WriteConsole(S+' : '+RealToStr(R,10)+' , '+formatdatetime('dd/mm/yyyy hh:mm:ss:zzz', R+1));
End;
```

See also: DateTime functions

MilliSecondsBetween

Returns the time in millisconds between two date/time

Format

Function **MilliSecondsBetween**(DataTime1,DataTime2:real):real;

Parameters

- DateTime1, DateTime2: Date/Time in real

Description

This function returns the number of milliseconds between two specified DateTime values.

MilliSecondsBetween always returns a positive result and therefore the parameter values are interchangeable.

Example

```
Var
  R,R1,R2 : Real;
Begin
  R1 := StrToDateTime('25/02/2018 15:25:23,456','dd/mm/yyyy hh:mm:ss:zzz');
  R2 := GetNow;
  R := MilliSecondsBetween(R1,R2);
  ShowMessage(IntToStr(R));
End;
```

See also: DateTime functions

GetNow

Returns the current Date/Time

Format

Function **GetNow**:real;

Description

The GetNow function returns the current date and time in the local time zone.

Example

```
Var
  R,R1,R2 : Real;
Begin
```

```
  R1 := StrToDateTime('25/02/2018 15:25:23,456','dd/mm/yyyy hh:mm:ss:zzz');
  R2 := GetNow;
  R := MilliSecondsBetween(R1,R2);
  ShowMessage(IntToStr(R));
End;
```

See also: DateTime functions

DDE

(Dynamic Data Exchange protocol)

Method

- App.DDE.check
- App.DDE.Execute
- App.DDE_Poke
- App.DDE.Request

App.DDE.Check

Check the presence of a DDE server

Format

Function **App.DDE.Check** (ServiceStr,TopicStr:String):boolean; // returns true if a DDE server is present

Parameters

- ServiceStr identify the DDE server (excel, winword, soffice…)
- TopicStr depends of the DDE server ([bookFileName_with_extension]SheetName, <FileName>, System, …)

Returned value

- True if a DDE link can be established

Description

Call App.DDE.check to verify if the couple ServiceStr/TopicStr is correct before using others DDE functions.

- ***Refer to the DDE manual for more details***

Example with Microsoft Word

```
Const
  CST_Service = 'winword';
  CST_Topic = 'DDE_VB.docm';
Var
  S:  String;
Begin
  if App.DDE.Check(CST_Service,CST_Topic) then
  Begin
    // In this example, a macro named Print_PDF must exit in the document (not in the normal.dotm)
    S := App.DDE.execute(CST_Service, CST_Topic,'[ThisDocument.Print_PDF("c:\temp\test.pdf")]',true);

    if length(S)>0 then ShowMessage(S);
    // the following line closes Word
    App.DDE.execute(CST_Service,'system','[AppClose]',true);
  End else ShowMessage('Error connecting to the DDE server');
End;
```

See also: DDE methods

App.DDE.Execute

Executes a macro in a DDE server

Format

Function **App.DDE.Execute** (ServiceStr,TopicStr, CmdStr:String; WaitFlag:boolean):String; // returns error message or empty string if success

Parameters

- ServiceStr identify the DDE server (excel, winword, soffice…)
- TopicStr depends of the DDE server ([bookFileName_with_extension]SheetName, <FileName>, System, …)
- CmdStr contain the macro to execute
- WaitFlag: specify if the client should wait or not

Description

Call App.DDE.Execute to send a single macro command to the server application. App.DDE.Execute returns an empty string if the macro was successfully passed to the DDE server application. If App.DDE.Execute was unable to send a command string, App.DDE.Execute returns a text error message.

CmdStr is a string that contains the macro to be executed by the DDE server application. The actual value of CmdStr depends on the DDE server application. See the documentation of the DDE server application for the command strings it will accept.

WaitFlag determines if this DDE client should wait until the DDE server application finishes executing the macro before allowing another DDE transaction to succeed. If WaitFlag is set to true, subsequent calls to App.DDE.Execute, App.DDE.Poke, App.DDE.Request will fail until the DDE server application completes the macro.

Attempting to execute a macro or poke data before a DDE server application completes a currently executing macro may cause the executing macro to fail or to produce unpredictable results. See the documentation of the DDE server application for the results of sending command strings or poking data before macro execution has completed.
Note: DDE_Execute returns an empty string if the macro command was successfully passed to the DDE server. A empty string does not ensure that the macro command will execute successfully once it has been accepted by the server.

- ***Refer to the DDE manual for more details***

Example with Microsoft Word

```
Var
  S:  String;
Begin
  // In this example, a macro named Print_PDF must exit in the document (not in the normal.dotm)
  S := dde_execute('winword', 'DDE_VB.docm','[ThisDocument.Print_PDF("c:\temp\test.pdf")]',true);

  if length(S)>0 then ShowMessage(S);
  // the following line closes Word
  dde_execute('winword','system','[AppClose]',true);
End;
```

Example with Microsoft Excel

```
Var
  S:  String;
Begin
  S := App.DDE.Execute('excel','DDE_VB.xls','[File.close()]',true); // Close the Sheet 'DDE_VB.xls'
  if length(S)>0 then ShowMessage(S);
  S := App.DDE.Execute('excel','system','[NEW(1)]',true); // Create a new sheet
  if length(S)>0 then ShowMessage(S);
End;
```

See also: DDE methods

App.DDE.Poke

Sends a data to a DDE server

Format

Function **App.DDE.Poke** (ServiceStr,TopicStr,ItemStr,DataStr:String):String; // returns error message or empty string if success

Parameters

- ServiceStr identify the DDE server (excel, winword, soffice…)
- TopicStr depends of the DDE server ([bookFileName_with_extension]SheetName, <FileName>, System, …)
- ItemStr specify the data to poke (cell reference, Bookmark etc…)
- DataStr contain the text to transfer

Description

Use App.DDE.Poke to transfer text data to a DDE server identified by ServiceStr and TopicStr that supports poked data.

ItemStr specifies the linked item in the DDE server. DataStr specifies the text data to transfer.

The value of the DDE item depends on the linked DDE server application. Item is typically a selectable portion of text, such as a spreadsheet cell or a database field in an edit box.

App.DDE.Poke returns an empty string if the data was successfully transferred, Text error message if the data was not successfully transferred.

- ***Refer to the DDE manual for more details***

Example with Microsoft Word

```
Var
  S:  String;
  n : integer;
Begin
  n := random(1000);
  S:= App.DDE.poke('winword','test.docx','MyBookmark',IntToStr(n));
  if length(S)>0 then ShowMessage(S);
End;
```

Example with Microsoft Excel

```
Var
  S:  String;
  n : integer;
Begin
  n := random(1000);
  S := App.DDE.poke('excel','[Book1.xls]Sheet1','R1C1',IntToStr(n));
  if length(S)>0 then ShowMessage(S);
End;
```

See also: DDE methods

App.DDE.Request

Requests data from a DDE server

Format

Function **App.DDE.Request** (ServiceStr,TopicStr,ItemStr:String):String; // returns the requested data or error message beginning by “ERROR:”..

Parameters

- ServiceStr identify the DDE server (excel, winword, soffice…)
- TopicStr depends of the DDE server ([bookFileName_with_extension]SheetName, <FileName>, System, …)
- ItemStr specify the data to request (cell reference etc…)

Description

Call App.DDE.Request to send a request to the DDE server for the data named by ItemStr. App.DDE.Request returns the value of the named DDE server item.

The value of the DDE item depends on the linked DDE server application. ItemStr is typically a selectable portion of text, such as a spreadsheet cell or a database field in an edit box.
Note: See the documentation for the DDE server application for specific information about specifying a DDE item.

- ***Refer to the DDE manual for more details***

Example with Microsoft Excel

```
Var
  S:  String;
Begin
  S := App.DDE.request('excel','[Book1.xls]Sheet1','R1C1');
  if length(S)>0 then ShowMessage(S);
End;
```

See also: DDE methods

Email

Method: SendEmail

SendEmail

Sends an email

Format

Function **SendEmail** (Address,Object,Message:String):boolean; // return true if the mail is enabled

Parameters

- Address: list of recipients (separator is semicolon)
- Object: object the email
- Message: Message of the email

Description

The function SendEmail allows sending email to one or more recipients.
The email option must be enabled in the options form otherwise the function won't have any effect.
Depending of the configuration, the sending of the email may be delayed. In the options check the parameters "Maximum delay before to send an email" and "Number of messages before to send an email".

Example

```
Var
  Msg : String;
Begin
  Msg := 'IMPORTANT'+#13;
  Msg := Msg+'Al cell temperature is ='+RealToStr(Growth.Al_ANB6_8_P6.temperature.MV,1);
  SendEmail('dest1@riber.fr;dest2@gmail.com','CrystalXE',Msg);
End;
```

Equipment

Methods

- PopupDetailView
- PopupSetupView
- PopupListView
- WriteTagValue
- WriteTagValueEx

(Equipment) WriteTagValue

Assigns a value to a tag identified by its id

Format

Procedure **WriteTagValue**(IdTag:word; Value:real; RunOnWrite:Boolean);

Parameters

- IdTag: The index of the tag (see GetTagId) (start at 0)
- Value: Value to set to the tag (the linearization will be applied)
- RunOnWrite: true to execute the event OnWrite of the Tag

Description

This procedure assigns a **value** to the tag identified by **IdTag** and executes the event OnWrite if required.
The reversed linearization using FxA and FxB is applied before to store the value into the table.
If the parameter SendToDevice is true, then the tag will be sent to the device.

To be send to the device, the tag must be of type R/W or Write.

If the tag identified by IdTag does not exist, an error message will be printed to the console.

Example

```
Var
  id: integer;
Begin
  id := GetTagId('MV',1);
  WriteTagValue(Id, 123, true);
End;
```

See also: Equipment methods

(Equipment) WriteTagValueEx

Assigns a value to a tag identified by its name and channel number

Format

Procedure **WriteTagValueEx**(TagName:string; Value:real; RunOnWrite:Boolean);

Parameters

- TagName: Name of the tag or Tag parameter
- Value: Value to set to the tag (the linearization will be applied)
- RunOnWrite: true to execute the event OnWrite of the Tag

Description

Assigns a **value** to the tag identified by **TagName** and execute the event OnWrite if required.
The reversed linearization using FxA and FxB is applied before to store the value into the table.
If the parameter SendToDevice is true, then the tag will be sent to the device.

The TagName can also be followed by:
.FxA
.FxB
.Min
.Max
.Frequency
Exemple of TagName: TSP.FxA will change the value of FxA

To be send to the device, the tag must be of type R/W or Write.

If the tag does not exist, an error message will be printed to the console.

Example

```
Begin
  WriteTagValueEx('SP', 123, true);
  WriteTagValueEx('MV', 1.23, false);
End;
```

See also: Equipment methods

PopupDetailView

Popup the detail view

Format

Procedure **PopupDetailView**(ShowModal:boolean);

Parameters

- ShowModal: Display the window as a modal form or not.

Description

Open the detail View of the equipment or the sub equipment in a floating window.
When a window is displayed modally, no input (keyboard or mouse click) can occur except to objects on the modal form.

Example

```
Begin
  Growth.Cell1.Temperature.PopupDetailView(false);
End;
```

See also: Equipment methods

PopupSetupView

Popup the setup view

Format

Procedure **PopupSetupView**(ShowModal:boolean);

Parameters

- ShowModal: Display the window as a modal form or not.

Description

Open the Setup View of the equipment or the sub equipment in a floating window.
When a window is displayed modally, no input (keyboard or mouse click) can occur except to objects on the modal form.

Example

```
Begin
  Growth.Cell1.Temperature.PopupSetupView(false);
End;
```

See also: Equipment methods

PopupListView

Popup the list view

Format

Procedure **PopupListView**(ShowModal:boolean);

Parameters

- ShowModal: Display the window as a modal form or not.

Description

Opens the list View of the equipment or the sub equipment in a floating window.
When a window is displayed modally, no input (keyboard or mouse click) can occur except to objects on the modal form.

Example

```
Begin
  Growth.Cell1.Temperature.PopupListView(false);
End;
```

See also: Equipment methods

File

Methods

- **INI Files:**
  ReadIniValue, WriteIniValue, ReadIniStr, WriteIniStr, SaveDataToIniFile, LoadDataFromIniFile
- **TEXT Files (CSV):**
  FClose, FLoadValue, FSaveValue, FLoadStr, FSaveStr, FRead, FReadLn, FWriteLn, Fseparator
- **Other File functions:**
  MakeDir, FileCat, DirExists, FileExists, FileOpen, FileCopy, FileDelete

ReadIniValue

Reads a value in an ini file

Format

Function **ReadIniValue**(Filename,Section,Key:string;default:real):real;

Parameters

- ***Filename***: Name of the ini file, the default search directory is the project data directory.
- ***Section***: the section in the ini file which is delimited by "[" and "]". To erase a section, the Key must be an empty string.
- ***Key***: name of the item in the section <key>=…, if the Key is an empty string then the section will be erased.
- ***Default***: default value if the item or section or file does not exist

Returned value

The value corresponding to the filename, section and key which has been read in the file or the default value if the item or the file was not found.

Description

This function reads an item in an INI file.
The .INI files has a text-based file format for representing application configuration data.
Initialization or Configuration Settings file (.INI) is a text file with 64Kb limit divided into sections, each containing zero or more keys. Each key contains zero or more values.
Example:

```
[SectionName]
keyname1=value
;comment
keyname2=value
```

Section names are enclosed in square brackets, and must begin at the beginning of a line. Section and key names are case-insensitive, and cannot contain spacing characters. The key name is followed by an equal sign ("="), optionally surrounded by spacing characters, which are ignored.

If the same section appears more than once in the same file, or if the same key appears more than once in the same section, then the last occurrence prevails.
A key can contain string, integer or boolean value.

Remark: If the Key name is an empty string then all the section will be cleared.

Example

```
begin
  WriteIniValue('test.ini','config','Name',567);
  ShowValue('Value=',readIniValue('test.ini','config','Name',123));// Value=567
end;
```

See also: All file functions

WriteIniValue

Writes a value in an ini file

Format

Procedure **WriteIniValue**(Filename,Section,Key:string; Value:real);

Parameters

- ***Filename***: Name of the ini file, the default search directory is the project data directory.
- ***Section***: the section in the ini file which is delimited by "[" and "]" To erase a section, the Key must be an empty string.
- ***Key***: name of the item in the section <key>=… If the Key is an empty string then the section will be erased.
- ***Value***: value to write

Description

This procedure writes an item in an INI file.
The .INI files have a text-based file format for representing application configuration data.
Initialization or Configuration Settings file (.INI) is a text file with 64Kb limit divided into sections, each containing zero or more keys. Each key contains zero or more values.
Example:

```
[SectionName]
keyname1=value
;comment
keyname2=value
```

Section names are enclosed in square brackets, and must begin at the beginning of a line. Section and key names are case-insensitive, and cannot contain spacing characters. The key name is followed by an equal sign ("="), optionally surrounded by spacing characters, which are ignored.

If the same section appears more than once in the same file, or if the same key appears more than once in the same section, then the last occurrence prevails.
A key can contain integer or real value.

Remark: If the Key name is an empty string then all the section will be cleared.

Example

```
begin
  WriteIniValue('test.ini','config','Name',567);
  ShowValue('Value=',readIniValue('test.ini','config','Name',123));// Value=567
end;
```

See also: All file functions

ReadIniStr

Reads a string in an ini file

Format

Function **ReadIniStr**(Filename,Section,Key,default : string):string;

Parameters

- ***Filename***: Name of the ini file, the default search directory is the project data directory.
- ***Section***: the section in the ini file which is delimited by "[" and "]"
- ***Key***: name of the item in the section <key>=…
- ***Default***: default string if the key or section or file does not exist

Returned value

The item corresponding to the filename, section and key which has been read in the file or the default string if the key or the file was not found.

Description

This function reads a string item in an INI file.
The .INI files has a text-based file format for representing application configuration data.
Initialization or Configuration Settings file (.INI) is a text file with 64Kb limit divided into sections, each containing zero or more keys.
Example:

```
[SectionName]
keyname1=string1
;comment
keyname2=string2
```

Section names are enclosed in square brackets, and must begin at the beginning of a line. Section and key names are case-insensitive, and cannot contain spacing characters. The key name is followed by an equal sign ("="), optionally surrounded by spacing characters, which are ignored.

If the same section appears more than once in the same file, or if the same key appears more than once in the same section, then the last occurrence prevails.

Example

```
begin
  WriteIniStr('test.ini','config','Name', 'Franck');
  ShowValue('Name=',readIniStr('test.ini','config','Name', 'Unknown'));// Name=Franck
end;
```

See also: All file functions

WriteIniStr

Writes a string character in an ini file

Format

Procedure **WriteIniStr**(Filename,Section,Key,Value:string);

Parameters

- ***Filename***: Name of the ini file, the default search directory is the project data directory.
- ***Section***: the section in the ini file which is delimited by "[" and "]"
- ***Key***: name of the item in the section <key>=…
- ***Value***: string character to write

Description

This procedure writes a string item in an INI file.
The .INI files have a text-based file format for representing application configuration data.
Initialization or Configuration Settings file (.INI) is a text file with 64Kb limit divided into sections, each containing zero or more keys. Each key contains zero or more values.
Example:

```
[SectionName]
keyname1=String 1
;comment
keyname2=String 2
```

Section names are enclosed in square brackets, and must begin at the beginning of a line. Section and key names are case-insensitive, and cannot contain spacing characters. The key name is followed by an equal sign ("="), optionally surrounded by spacing characters, which are ignored.

If the same section appears more than once in the same file, or if the same key appears more than once in the same section, then the last occurrence prevails.

Example

```
begin
  WriteIniStr('test.ini','config','Name', 'Franck');
  ShowValue('Name:',readIniStr('test.ini','config','Name', 'Unknown'));// Name:Franck
end;
```

See also: All file functions

FOpen

Opens a CSV file for reading or writing

Format

Function **FOpen**(id:byte; FileName:string; mode:byte):Boolean;

Parameters

- ***Id***: The file identifier (1 to 10) to be used with all other file functions beginning by F...
- ***FileName***: Full file name to read
- ***mode***=fm_Write(0), fm_Read(1), fm_WriteVerify(2), fm_Append(3)

Returned value

- True if the file has opened, False if an error occurred.

Description

The function FOpen opens a file in the following modes:
- Writing without condition: fm_Write(0)
- Reading: fm_Read(1)
- Writing with a message to the operator if the file already exists: fm_WriteVerify(2)
- Append: fm_Append(3)

The default directory is the data sub directory in the project folder but a full path can be defined.

Example

```
Var
  Ok : boolean;
  i  : integer;
begin
  ok := Fopen(1,'test.csv',fm_Write);
  if ok then
  Begin
    // Save the title line
    for i:=1 to 12 do FSaveStr(1,'C'+IntToStr(i));
    FWriteln(1);
    // Save the first record line
    for i:=1 to 12 do FSaveValue(1,i);
    FWriteln(1);
    // Save the second record line
    for i:=1 to 12 do FSaveValue(1,i);
    FWriteln(1);
  end else ShowMessage('Error writing the file');
  Fclose(1);
end;
```

See also: All file functions

FClose

Closes a CSV file

Format

Procedure **FClose**(id:byte);

Parameters

- ***Id***: file identifier used by the function Fopen

Description

This procedure closes the file after opening by Fopen

Example

```
Var
  Ok : boolean;
  i  : integer;
begin
  ok := Fopen(1,'test.csv',fm_Write);
  if ok then
  Begin
    // Save the title line
    for i:=1 to 12 do FSaveStr(1,'C'+IntToStr(i));
    FWriteln(1);
    // Save the first record line
    for i:=1 to 12 do FSaveValue(1,i);
    FWriteln(1);
    // Save the second record line
    for i:=1 to 12 do FSaveValue(1,i);
    FWriteln(1);
  end else ShowMessage('Error writing the file');
  Fclose(1);
end;
```

See also: All file functions

FLoadValue

Loads a real number from a CSV file

Format

Function **FLoadValue**(id:byte):real

Parameters

- ***Id***: file identifier

Returned value

- The value read in the file at the current position

Description
The Function FLoadValue reads a value in the file identified by **id**.
The file must be opened by the function FOpen with the same identifier.
The value is read and returned as a real number. The current position in the file is moved to the position after the next separator or to the next line.

If an error occurred (the value cannot be converted to a real number, or the file is not readable or not opened, or the end of file is reached) then the function will return -999999

If the value begins by '$' then the function will convert a hexadecimal value.

Example

```
Var
  Ok : boolean;
begin
  ok := Fopen(1,'test.csv',fm_Read);
  if ok then
  Begin
    ShowValue('Value1=',FLoadValue(1));
    ShowValue('Value2=',FLoadValue(1));
    ShowValue('Value3=',FLoadValue(1));
    ShowValue('Value4=',FLoadValue(1));
  end else ShowMessage('Error reading the file');
  Fclose(1);
end;
```

See also: All file functions

FSaveValue

Saves a real number in a CSV file

Format

Procedure **FSaveValue**(id:byte; Value:Real);

Parameters

- ***Id***: file identifier
- ***Value***: real number to save in the file specified by Id

Description

This procedure saves a value in a CSV file in append mode followed by a separator.
By default the separator is the semicolon character (";") but it can be redefined with the function FSeparator.

Example

```
Var
  Ok : boolean;
  i  : integer;
begin
  ok := Fopen(1,'test.csv',fm_Write);
  if ok then
  Begin
    // Save the title line
    for i:=1 to 12 do FSaveStr(1,'C'+IntToStr(i));
    FWriteln(1);
    // Save the first record line
    for i:=1 to 12 do FSaveValue(1,i);
    FWriteln(1);
    // Save the second record line
```

```
    for i:=1 to 12 do FSaveValue(1,i);
    FWriteln(1);
  end else ShowMessage('Error writing the file');
  Fclose(1);
end;
```

See also: All file functions

FLoadStr

Reads a string in a file

Format

Function **FLoadStr**(id:byte):string;

Parameters

- ***Id***: file identifier

Returned value

- The string read in the file at the current position

Description

The Function FLoadStr reads a string in the file identified by **id** and delimited by the separator (semicolon by default) the current position in the file is moved to the position after the next separator or to the next line.

The file must be opened by the function FOpen with the same identifier.

If an error occurred (the file is not readable or not opened, or the end of file is reached) then the function will return the string '_ERR_'

Example

```
Var
  Ok : boolean;
begin
  ok := Fopen(1,'test.csv',fm_Read);
  if ok then
  Begin
    ShowMessage('Value1='+FLoadStr(1));
    ShowMessage('Value2='+FLoadStr(1));
    ShowMessage('Value3='+FLoadStr(1));
    ShowMessage('Value4='+FLoadStr(1));
    ShowMessage('Value5='+FLoadStr(1));
  end else ShowMessage('Error reading the file');
  Fclose(1);
end;
```

See also: All file functions

FSaveStr

Saves a text line in a file

Format

Procedure **FSaveStr**(id:byte; S:string);

Parameters

- ***Id***: file identifier

Description

This procedure saves a string in a CSV file in append mode followed by a separator.
By default the separator is the semicolon character (";") but it can be redefined with the function FSeparator.

Example

```
Var
  Ok : boolean;
  i  : integer;
begin
  ok := Fopen(1,'test.csv',fm_Write);
  if ok then
  Begin
    // Save the title line
    for i:=1 to 12 do FSaveStr(1,'C'+IntToStr(i));
    FWriteln(1);
    // Save the first record line
    for i:=1 to 12 do FSaveValue(1,i);
    FWriteln(1);
    // Save the second record line
    for i:=1 to 12 do FSaveValue(1,i);
    FWriteln(1);
  end else ShowMessage('Error writing the file');
  Fclose(1);
end;
```

See also: All file functions

FReadLn

Reads a text line in a file

Format

Function **FReadLn**(id:byte):string;

Parameters

- Id: file identifier

Returned value

- The text line of the current line in the file identified by id.

Description

The Function FReadln reads a string in the file identified by **id** and delimited by the end of line (CR/LF).
The current position in the file is moved to the next line.

The file must be opened by the function FOpen with the same identifier.

If an error occurred (the file is not readable or not opened, or the end of file is reached) then the function will return the string '_ERR_'

Example

```
Var
  Ok : boolean;
begin
  ok := Fopen(1,'test.csv',fm_Read);
  if ok then
  Begin
    ShowMessage('Line #1='+FReadLn(1));
    ShowMessage('Line #2='+FReadLn(1));
    ShowMessage('Line #3='+FReadLn(1));
    ShowMessage('Line #4='+FReadLn(1));
    ShowMessage('Line #5='+FReadLn(1));
  end else ShowMessage('Error reading the file');
  Fclose(1);
end;
```

See also: All file functions

FRead

Reads a value composed of 1 to 8 bytes in a file

Format

Function **FRead**(id:byte; count:byte):int64;

Parameters

- Id: file identifier
- Count: number of byte to read (1 to 8)

Returned value

- The value, built with count bytes, read in the file at the current position.

Description

This function reads count bytes in the file identified by the id at the current position and increment the position pointer to the next value. The next value is separated by the column separator. By default the separator is the semicolon but it can be changed with the procedure Fseparator.

The order in which the sequence of bytes is read in the file is little endian, so the value in the file must begin by least significant bytes (LSB).

If an error occurred (the file is not readable or not opened, or the end of file is reached) then all returned bytes will be set to $FF (255)

Example 1:
The file contains the following bytes: $31 $09 $30 $78
After opening the file, the first call of the function Fread(1,2) will return the number $931 (2 353 in decimal)
The second call of the function Fread(1,2) will return the number $7830 (30 768 in decimal)

And also, after opening the file, the first call of the function Fread(1,4) will return the number $78300931 (2 016 414 001)

Example

```
Var
  Ok: boolean;
  R : real;
begin
  ok := Fopen(1,'test.csv',fm_Read);
  if ok then
  Begin
    R := Fread(1,4);
    ShowMessage('Value= $'+IntToHex(R,8)+' ('+IntToStr(R)+')');
    R := Fread(1,4);
    ShowMessage('Value= $'+IntToHex(R,4)+' ('+IntToStr(R)+')');
    R := Fread(1,1);
    ShowMessage('Value= $'+IntToHex(R,1)+' ('+IntToStr(R)+')');
  end else ShowMessage('Error reading the file');
  Fclose(1);
end;
```

See also: All file functions

FWriteLn

Add a new empty line in a text file

Format

Procedure **FWriteln**(id:byte);

Parameters

- Id: file identifier

Description

This procedure adds a new line in the file.
This procedure will append a carriage return and a line feed characters in the file identified by the id.

Example

```
Var
  Ok : boolean;
  i  : integer;
begin
  ok := Fopen(1,'test.csv',fm_Write);
  if ok then
  Begin
    // Save the title line
    for i:=1 to 12 do FSaveStr(1,'C'+IntToStr(i));
    FWriteln(1);
    // Save the first record line
    for i:=1 to 12 do FSaveValue(1,i);
    FWriteln(1);
    // Save the second record line
    for i:=1 to 12 do FSaveValue(1,i);
    FWriteln(1);
  end else ShowMessage('Error writing the file');
  Fclose(1);
end;
```

See also: All file functions

FSeparator

Defines the separator between two columns in a CSV file

Format

Procedure **FSeparator**(sep:Char);

Parameters

- sep: define the separator for function FSaveValue and FSaveStr

Description

By default the separator between two column is the semicolon character (";") but it can be redefined with this function.

Example

```
Var
  Ok : boolean;
  i  : integer;
Begin
  Fseparator(#9); // define the tabulation character as column separator
  ok := Fopen(1,'test.csv',fm_Write);
  if ok then
  Begin
    // Save the title line
    for i:=1 to 12 do FSaveStr(1,'C'+IntToStr(i));
    FWriteln(1);
    // Save the first record line
    for i:=1 to 12 do FSaveValue(1,i);
    FWriteln(1);
    // Save the second record line
    for i:=1 to 12 do FSaveValue(1,i);
    FWriteln(1);
  end else ShowMessage('Error writing the file');
  Fclose(1);
end;
```

See also: All file functions

MakeDir

Creates a directory

Format

Function **MakeDir**(dir:string):boolean;

Parameters

- dir: directory to create

Returned value

- True if the directory has been created with success.

Description

Create a directory identified by the string **dir**.
Several directories can be created in the same time (ex: /Dir1/Dir2/Dir3)
If the full path is not given, then the directory will be created into the data directory of the current project.

Examples:

- *MakeDir('\TestDir')* or *MakeDir('TestDir')* will create the subdirectory "Testdir" in the data directory of the current project.
- *MakeDir('Users/User1/John')* will create the subdirectory "*Users/User1/John*" in the data directory of the current project.
- *MakeDir('c:\TestDir')* will create the directory "c:\Testdir".

Example

```
begin
  if not(MakeDir('TestDir'))
    then ShowMessage('Error creating the directory');
    else ShowMessage('Success creating directory');
end;
```

See also: All file functions

FileCat

Concatenates two files

Format

Function **FileCat**(dest,source1,source2:string):Boolean;

Parameters

- ***Dest***: destination file name
- ***Source1***: first file
- ***Source2***: second file

Returned value

- True if the file has been successfully created

Description

The function FileCat concatenates the files identified by **source1** and **source2** into one identified by **dest**.
If the function failed (source file not found or error writing the destination file) then the function returns false else the function returns true.

Example

```
Begin
  if not(FileCat('test.cat','test.csv','test.ini'))
    then ShowMessage('Error concatenating the files');
    else ShowMessage('Success concatenating the files');
end;
```

See also: All file functions

DirExists

Returns true if the given directory exists

Format

Function **DirExists**(DirName : string):Boolean;

Parameters

- ***DirName***: name of the directory to check

Returned value

- True if the directory exists

Description

The DirExists function returns True if the given DirName file exists.
If no drive letter is specified and no network directory (if the name does not begin by "\\") is specified, then the directory is searched for in the data directory of the current project directory.
If the directory name begins by '\', the directory will be searched for in the project directory.

False may be returned if the user is not authorised to see the file.

Example

```
begin
  if DirExists('testDir')
    then ShowMessage('Yes, the directory exists in the data directory of this project');
    else ShowMessage('No, the directory does not exist');
end;
```

See also: All file functions

FileExists

Returns true if the given file exists

Format

Function **FileExists**(FileName : string):Boolean;

Parameters

- ***FileName***: path + name of the file to check

Returned value

- True if the file exists

Description

The FileExists function returns True if the given File name exists.

The FileName can be preceed by a path.
If no drive letter is specified and no network directory is specified (if the name does not begin by "\\"), then the directory is searched from the project directory.

False may be returned if the user is not authorised to see the file.

Example

```
Begin
  if FileExists('recipe\growth1\vcsel.rcp')
    then ShowMessage('Yes, the file has been found in this project');
    else ShowMessage('No, the file does not exist');
end;
```

See also: All file functions

FileOpen

Open a form (not to be confused with FOpen)

Format

Function **FileOpen**(FName,Params:string; mode:int32) : integer;

Parameters

- ***FName***: File name to open
- ***Params***: parameters of the file to open (available for forms)
- ***Mode***: Modal or not modal

Returned value

- Returns -1 if the function failed (example: if the file does not exist)
- Returns 0 if no modal and file is opened
- Returns the modal result value if it is a modal (-2 if the user close the window by the cross)

Description

The FileOpen function can be used to open a form.

If no extension is provided then the file is supposed to be a form.
Params will be assigned to the property "params" of the form and can be used in the visual objects and scripts of the forms.

Mode = 1 to open the form in modal mode or 0 for non modal.
A modal form is one that has to be dealt with before a user can continue and have access to other forms.

The form can use the function CloseModal(x) to return the value specified by x
By example, the form can contain two buttons (Ok and cancel) with a script executing the function CloseModal when clicking on it. The button Ok will return 1 and the button cancel will return 2.

Example

```
Var
  i : integer;
begin
  i := Fileopen('test','Hello',1);
  ShowValue('result=',i);
end;
```

See also: All file functions

FileCopy

Copies an existing file to a new file

Format

Function **FileCopy**(SrcName, DestName : string):Boolean;

Parameters

- ***SrcName***: Name of the file to copy with the full path and extension
- ***DestName***: Name of the new file with the full path and extension

Returned value

- True if the file has been successfully copied.

Description

This function copies the file ***SrcName*** into a new file with the name ***DestName***.
If the function failed (if the file does not exist, or if the file is read only or if the file is openned) then the function returns false else the function returns true.

Example

```
begin
```

```
  if not(FileCopy('d:\riber\Project\data\mydatafile.txt',
'd:\riber\Project\data\mycopydatafile.txt'))
    then ShowMessage('Error copying the file');
    else ShowMessage('Success copying the file');
end;
```

See also: All file functions

FileDelete

Deletes a file

Format

Function **FileDelete**(Filename:string):Boolean;

Parameters

- ***Filename***: Name of the file to delete with the full path and extension

Returned value

- True if the file existed and if it has been successfully delete.

Description

This function deletes the file with the name given in parameter (***filename).***
If the function failed (the file does not exist, or the file is read only or the file is openned) then the function returns false else the function returns true.

Example

```
begin
  if not(FileDelete('d:\riber\Project\data\mydatafile.txt'))
    then ShowMessage('Error deleting the file');
    else ShowMessage('Success deleting the file');
end;
```

See also: All file functions

Forms

Methods

- Close,
- LoadForeGroundFromFile,
- Print,
- CopyToClipBoard,
- SaveImageToFile,
- CloseModal,
- SaveDataToIniFile,
- LoadDataFromIniFile,
- CreateObject

Close

(Form method)

Closes the form

Format

Procedure **Close**;

Description

Closes the form.
If this procedure is called in a script of the form which is closed then the next instruction will never be executed.

Example
```
Begin
  Close;
End;
```

See also: Forms methods

LoadForeGroundFromFile
(Form method)

Loads a picture

Format

Function **LoadForeGroundFromFile**(Filename:String):Boolean

Parameters

Filename: File name of the picture file (the default directory is the project form directory)

Returned value

True if succeeded

Description

Loads a foreground picture from a bitmap file (.bmp)
The default search directory is the project form directory.

Example
```
Begin
  LoadForeGroundFromFile('synoptic.bmp');
End;
```

See also: Forms methods

Print
(Form method)

Prints the form

Format
procedure **print**(AskPrinter : boolean)

Parameters

AskPrinter: if true, open the printer dialog box, else the default printer is used.

Description

This procedure prints the form and Ask for the printer or not.

Example
```
Begin
  Print(true);
End;
```

See also: Forms methods

CopyToClipBoard
(Form method)

Copies the form to the clipboard

Format

procedure **CopyToClipBoard**;

Description

This function copies the form to the clipboard.

Example
```
Begin
  CopyToClipBoard;
End;
```

See also: Forms methods

SaveImageToFile
(Form method)

Saves an image of the form to a file

Format

Procedure **SaveImageToFile**(Filename:string)

Parameters

FileName: Output file name

Description

Saves an image of the form in a file (BMP file)
The default extension is .BMP
The default directory is the data directory

Example
```
Begin
  SaveImageToFile('MyCopy.bmp');
End;
```

See also: Forms methods

CloseModal
(Form method)

Closes the form with a returned value

Format

Procedure **CloseModal**(ReturnedValue:integer)

Parameters

ReturnedValue: integer number

Description

Closes the form and return a number. This is usefull for modal form to return a value when the user push on OK or Cancel.
Standard value for Ok is 1 and Cancel is 2.
See also FileOpen for more details.

Example
```
Begin
  CloseModal(2);
End;
```

See also: Forms methods

SaveDataToIniFile

(Form method)

Saves user data of some objects to ini file

Format

Procedure **SaveDataToIniFile**(Filename:string)

Parameters

Filename: Output file name

Description

Saves the content of some objects to an ini file.
Data which is saved depends of the type of the object.
For an indicator, the data is the input value.
For a checkbox, the data is the status checked or not.
For a radio button, the data is the status checked or not.
For a comboBox, the data is the selected item index
For an animated image, the data is the selected image index

The file can be read with the function LoadDataFromIniFile

The default extension is .BMP
The default directory is the data directory

Example

```
Begin
  SaveDataToIniFile('Myconfig.ini');
End;
```

See also: Forms methods, file functions

LoadDataFromIniFile

(Form method)

Loads user data of some objects to ini file

Format

Function **LoadDataFromIniFile**(Filename:string):Boolean

Parameters

Filename: Input file name

Description

Loads the content of some objects from an ini file.
Data which is loaded depends of the type of the object.
For an indicator, the data is the input value.
For a checkbox, the data is the status checked or not.
For a radio button, the data is the status checked or not.
For a comboBox, the data is the selected item index
For an animated image, the data is the selected image index

The file can be read with the function LoadDataFromIniFile

The default extension is .BMP
The default directory is the data directory

Example
```
Begin
  LoadDataFromIniFile('Myconfig.ini');
End;
```

See also: Forms methods, file functions

CreateObject
(Form method)

Creates a new object in the form

Format

function **CreateObject**(ClassName:String;x,y:integer):Name

Parameters

ClassName: Class name of the object to create
x,y: initial position of the object in the form

Returned value

Name of the new object

Description

Creates a new object in the form at the position **x** and **y**.

Exemple of object class:
TObjIndicator
TObjBitBtn
TObjBtnConfirm
TObjCamera
TObjCheckbox
TObjCombo
TMyGroupBox
TObjImage
TObjImageMultiple
TMyCustomLabel
TObjLight
TObjProgressBar
TObjRadio

Example
```
Begin
  ObjectName := CreateObject('TObjIndicator',100,30);
  SetPropStr(ObjectName+'.Caption','Essai');
End;
```

See also: Forms methods

Logical

Methods
- CheckBit
- ChgBit
- ClearBit
- Not
- NotBit
- SetBit
- Shl
- Shr

CheckBit

Checks a bit

Format

Function **CheckBit**(value: dword; BitNum:byte):Boolean;

Parameters

- Value: value in which to test one bit.
- BitNum: number of bit to check, starting at 0, its the lowest significative bit.

Returned value

- True if the bit is equal to 1 else false

Description

The CheckBit function is used to check the status of one bit into a value.

Example

```
Var
  num1 : Word;
begin
  num1 := 2;
  if CheckBit(num1,1)
   then ShowMessage('true');
   else ShowMessage('false');
end;
```

See also: Logical functions

ChgBit

Change a bit with the status passed in parameter

Format

Function **ChgBit**(value: dword; bitNum:byte; Status:boolean): dword;

Parameters

- Value: source value
- BitNum: number of bit to clear starting at zero
- Status: Boolean;

Returned value

- The new value with the bit changed

Description

This function allows to change the state of a bit in a 32 bits value. The state is passed in parameter.

Example

```
var
  Val : Word;
begin
  Val := $3C;
  ShowMessage('Before ChgBit: $'+IntToHex(Val,4)); // Before ClearBit: $003C
  Val := ChgBit(Val,4,0);
  ShowMessage('After ChgBit: $'+IntToHex(Val,4)); // After ClearBit: $002C
end;
```

See also: Logical functions

ClearBit

Clears a bit

Format

Function **ClearBit**(value: dword;bitNum:byte): dword;

Parameters

- Value: source value
- BitNum: number of bit to clear starting at zero

Returned value

- The new value with the bit changed

Description

The ClearBit function is used to check the status of one bit into a value.

Example

```
var
  Val : Word;
begin
  Val := $3C;
  ShowMessage('Before ClearBit: $'+IntToHex(Val,4)); // Before ClearBit: $003C
  Val := ClearBit(Val,4);
  ShowMessage('After ClearBit: $'+IntToHex(Val,4)); // After ClearBit: $002C
end;
```

See also: Logical functions

Not

Boolean Not of one arguments

Format

Function **Not**(condition:Boolean):Boolean;

Description

Performs boolean 'Not' of a logical value. If True, the value becomes False. If False, the value becomes True.

Note: To perform a bitwise not, refer to the function notbit.

Example

```
Var
  num1 : Word;
begin
  num1 := 2;
  if Not (num1 > 0)
    then ShowMessage('num1 <= 0');
    else ShowMessage('num1 > 0');
end;
```

See also: Logical functions

NotBit

Bitwise not

Format

Function **Notbit**(value: dword): dword;

Parameters

- Value: source value

Returned value

- True if the bit is equal to 1 else false

Description

The CheckBit function is used to check the status of one bit into a value.

Example

```
var
  Val : Word;
begin
  Val := $3C;
  ShowMessage('Before NotBit: $'+IntToHex(Val,4)); // Before NotBit: $003C
  Val := NotBit(Val);
  ShowMessage('After NotBit: $'+IntToHex(Val,4)); // After NotBit: $FFC3
end;
```

See also: Logical functions

SetBit

Sets a bit

Format

Function **SetBit**(value: dword;bitNum:byte): dword;

Parameters

- Value: source value
- BitNum: number of bit to turn on starting at zero

Returned value

- The new value with the bit changed

Description

The SetBit function is used to check the status of one bit into a value.

Example

```
var
  Val : Word;
begin
  Val := $3C;
  ShowMessage('Before SetBit: $'+IntToHex(Val,4)); // Before SetBit: $003C
  Val := SetBit(Val,8);
  ShowMessage('After SetBit: $'+IntToHex(Val,4)); // After SetBit: $013C
end;
```

See also: Logical functions

Shl

Shift an integer value left by a number of bits

Format

Function **Shl**(value: dword;bitCnt:byte): dword;

Parameters

- Value: source value to Shift.
- BitCnt: number of bits to shift.

Returned value

- The new value shifted

Description

The Shl function performs a bitwise shift left of an Integer. The number is shifted Bits to the left.

Example

```
var
  before, after : Word;
begin
  // Set up out starting number
  before := $3C;      // Hex 3C = 003C in the Word
  ShowMessage('Before $'+IntToHex(before,2));

  // Shift left by 12 will lose the top 12 bits of the Word
  after := Shl(before,12);
  ShowMessage('After shift left of 12 bits $'+IntToHex(after,2));

  // Shifting right by 12 will not recover the lost data
  after := Shr(after,12);
  ShowMessage('After shift right of 12 bits (will not recover the lost data) $'+IntToHex(after,2));
end;
```

See also: Logical functions

Shr

Shift an integer value right by a number of bits

Format

Function **Shr**(value: dword;bitCnt:byte): dword;

Parameters

- Value: source value to Shift.
- BitCnt: number of bits to shift.

Returned value

- The new value shifted

Description

The Shr function performs a bitwise shift right of an Integer. The number is shifted Bits to the right.

Example

```
var
  before, after : Word;
begin
  // Set up out starting number
  before := $3C;      // Hex 3C = 003C in the Word
  ShowMessage('Before $'+IntToHex(before,2));

  // Shift left by 12 will lose the top 12 bits of the Word
  after := Shl(before,12);
  ShowMessage('After shift left of 12 bits $'+IntToHex(after,2));

  // Shifting right by 12 will not recover the lost data
  after := Shr(after,12);
  ShowMessage('After shift right of 12 bits (will not recover the lost data) $'+IntToHex(after,2));
end;
```

See also: Logical functions

Mathematical

Methods

Abs

Gives the absolute value of a number

Format

Function **Abs**(Value: real):real;

Description

The Abs function returns the absolute value of a negative or positive number. It does this by removing a negative sign, if found.

Example

```
var
  f, bigFloat : real;
  i : Integer;
begin
  f    := -1.5;       // Small negative floating point number
  bigFloat := -4.56E100;  // Large negative floating point number
  i      := -7;         // Negative integer

  ShowValue('Abs(f) = ',Abs(f)); // =1.5
  ShowValue('Abs(bigFloat) = ',Abs(bigFloat)); // =4.56E100
  ShowValue('Abs(i) = ',Abs(i)); // =7
end;
```

See also: Mathematical methods

ArcTan

The Arc Tangent of a number, returned in radians

Format

Function **ArcTan**(Value: real):real;

Description

The ArcTan function is a mathematical function giving the value in radians of the Arc Tangent of Number.
PI radians = 180 degrees

Example

```
var
  f : real;
begin
  // The ArcTan of PI/2 should give 1.0
  f := ArcTan(PI/2);
```

```
  ShowValue('ArcTan of PI/2 = ', f);
end;
```

See also: Mathematical methods

Ceil

Rounds the number up to the nearest integer

Format

Function Ceil(Value: real):int32;

Description

Returns as an integer 32 bits the nearest integer that is equal to or greater than the given Value.

Example

```
begin
  ShowValue('Round(12.75) = ',Round(12.75)); // Round(12.75) =  13
  ShowValue('Round(-12.75)= ',Round(-12.75));// Round(-12.75)= -13
  ShowValue('Trunc(12.75)= ' ,Trunc(12.75)); // Trunc(12.75) =  12
  ShowValue('Trunc(-12.75)= ',Trunc(-12.75));// Trunc(-12.75)= -12
  ShowValue('Int(12.75)= '   ,Int(12.75));   // Int(12.75)   =  12
  ShowValue('Int(-12.75)= '  ,Int(-12.75));  // Int(-12.75)  = -12
  ShowValue('Frac(12.75)= '  ,Frac(12.75));  // Frac(12.75)  = 0.75
  ShowValue('Frac(-12.75)= ' ,Frac(-12.75)); // Frac(-12.75) =-0.75
  ShowValue('Ceil(12.75)= '  ,Ceil(12.75));  // Ceil(12.75)  =  13
  ShowValue('Ceil(-12.75)= ' ,Ceil(-12.75)); // Ceil(-12.75) = -12
  ShowValue('Floor(12.75)= ' ,Floor(12.75)); // Floor(12.75) =  12
  ShowValue('Floor(-12.75)= ',Floor(-12.75));// Floor(-12.75)= -13
end;
```

See also: Mathematical methods

Cos

The cosine of a number

Format

Function Cos(Value: real):real;

Description

The Cos function is a mathematical function giving the Cosine value of a Number value given radians.
PI radians = 180 degrees

Example

```
var
  f : real;
begin
  // The Cosine of 60 degrees = 0.5
  f := Cos(PI/3);   // = 180/3 = 60 degrees
  ShowValue('Cos(PI/3) = ', f);
end;
```

See also: Mathematical methods

Exp

Gives the exponent of a number

Format

Function **Exp**(Value: real):real;

Description

The Exp function takes an integer or floating point Number and raises e (2.72) to that power.
This is only used for mathematical processing.
Exp has the opposite effect to Ln - the natural logarithm.

Example

```
var
  f : Real;
begin
  // Get the natural logarithm of 2
  f := Ln(2);

  // Show this value (0.69314...)
  ShowValue('Ln(2) = ',f);

  // Get the exponent of this value - reverses the Ln operation
  f := Exp(f);

  // Show this value (2)
  ShowValue('Exp(Ln(2)) = ',f);
end;
```

See also: Mathematical methods

Floor

Rounds the number down to the nearest integer

Format

Function Ceil(Value: real):int32;

Description

Returns as an integer 32 bits the nearest integer that is equal to or greater than the given Value.

Example

```
begin
  ShowValue('Round(12.75) = ',Round(12.75)); // Round(12.75) =  13
  ShowValue('Round(-12.75)= ',Round(-12.75));// Round(-12.75)= -13
  ShowValue('Trunc(12.75)= ' ,Trunc(12.75)); // Trunc(12.75) =  12
  ShowValue('Trunc(-12.75)= ',Trunc(-12.75));// Trunc(-12.75)= -12
  ShowValue('Int(12.75)= '   ,Int(12.75));   // Int(12.75)   =  12
  ShowValue('Int(-12.75)= '  ,Int(-12.75));  // Int(-12.75)  = -12
  ShowValue('Frac(12.75)= '  ,Frac(12.75));  // Frac(12.75)  = 0.75
  ShowValue('Frac(-12.75)= ' ,Frac(-12.75)); // Frac(-12.75) =-0.75
  ShowValue('Ceil(12.75)= '  ,Ceil(12.75));  // Ceil(12.75)  =  13
  ShowValue('Ceil(-12.75)= ' ,Ceil(-12.75)); // Ceil(-12.75) = -12
  ShowValue('Floor(12.75)= ' ,Floor(12.75)); // Floor(12.75) =  12
  ShowValue('Floor(-12.75)= ',Floor(-12.75));// Floor(-12.75)= -13
end;
```

See also: Mathematical methods

Frac

Returns the fractional part of a floating point number

Format

Function **Frac**(Value: real):real;

Description

The Frac function returns the fractional part of a floating point number.

Example

```
begin
```

```
  ShowValue('Round(12.75) = ',Round(12.75)); // Round(12.75) =  13
  ShowValue('Round(-12.75)= ',Round(-12.75));// Round(-12.75)= -13
  ShowValue('Trunc(12.75)= ' ,Trunc(12.75)); // Trunc(12.75) =  12
  ShowValue('Trunc(-12.75)= ',Trunc(-12.75));// Trunc(-12.75)= -12
  ShowValue('Int(12.75)= '   ,Int(12.75));   // Int(12.75)   =  12
  ShowValue('Int(-12.75)= '  ,Int(-12.75));  // Int(-12.75)  = -12
  ShowValue('Frac(12.75)= '  ,Frac(12.75));  // Frac(12.75)  = 0.75
  ShowValue('Frac(-12.75)= ' ,Frac(-12.75)); // Frac(-12.75) =-0.75
  ShowValue('Ceil(12.75)= '  ,Ceil(12.75));  // Ceil(12.75)  =  13
  ShowValue('Ceil(-12.75)= ' ,Ceil(-12.75)); // Ceil(-12.75) = -12
  ShowValue('Floor(12.75)= ' ,Floor(12.75)); // Floor(12.75) =  12
  ShowValue('Floor(-12.75)= ',Floor(-12.75));// Floor(-12.75)= -13
end;
```

See also: Mathematical methods

Int

Same as Frac

See also: Mathematical methods

Ln

Gives the natural logarithm of a number

Format

Function **Ln**(Value: real):real;

Description

he Ln function takes an integer or floating point Number and gives the natural logarithm of that number.
This is only used for mathematical processing.
Ln has the opposite effect to Exp - the exponent of a number. (2.72 raised to that number power)

Example

```
var
  float : Double;
begin
  // Get the natural logarithm of 2
  float := Ln(2);

  // Show this value
  ShowMessage('Ln(2) = '+RealToStr(float)); // Ln(2) = 0.693147180559945

  // Get the exponent of this value - reverses the Ln operation
  float := Exp(float);

  // Show this value
  ShowMessage('Exp(Ln(2)) = '+RealToStr(float));// Exp(Ln(2)) = 2
end;
```

See also: Mathematical methods

Log

Gives the Logarithm of a number

Format

Function **Log**(Value: real):real;

Description

Returns the Natural Logarithm (that is, to base E) of Value, or the logarithm to the given Base.

Example

```
var
  result : real;
begin
  // Log of 10 in base E = 2.30258509299405
  result := Log(10);
  ShowValue('Log(10)    = {0}', result);
End;
```

See also: Mathematical methods

Pi

PI constant

Format

Function **Pi**:real;

Description

The Pi function returns a floating point value giving a useful approximation of the value of Pi.
The circumference of a circle Pi * Diameter.

Example

```
var
  myPi : Real;
  diam : Integer;
begin
  diam := 10;

  // Show circumference and the area of a circle
  // with diameter - 10 cm
  ShowValue('     Diameter = '+Diam);// Diameter=10
  ShowValue ('Circumference = '+Pi*Diam); // Circumference = 31.41592
  ShowValue ('         Area = '+Pi*(diam/2)*(diam/2)); // Area =78.539
end;
```

See also: Mathematical methods

Random

Generates a random floating point or integer number

Format

Function **Random**(range: int32):real;

Description

The Random function generates random numbers in the range : 0 <= Number < Range
Crystal uses a pseudo random number generator that always returns the same sequence of values (2e32) each time the program runs.

Example

```
var
  float : real;
  V   : Integer;
  i     : Integer;
begin
  // Get an integer random number in the range 1..100
  for i := 1 to 5 do
  begin
    V := 1 + Random(100);    // The 100 value gives a range 0..99
    ShowMessage('int = '+IntToStr(V));
  end;
end;
```

See also: Mathematical methods

Round

Rounds a floating point number to an integer

Format

Function Round(Value: real):int32;

Description

The Round function rounds a floating point Number to an Integer value.
The rounding uses Bankers rules, where an exact half value causes a rounding to an even number:

12.4 rounds to 12
12.5 rounds to 12 // Round down to even
12.6 rounds to 13
13.4 rounds to 13
13.5 rounds to 14 // Round up to even
13.6 rounds to 14

Example

```
begin
  ShowValue('Round(12.75) = ',Round(12.75)); // Round(12.75) =  13
  ShowValue('Round(-12.75)= ',Round(-12.75));// Round(-12.75)= -13
  ShowValue('Trunc(12.75)= ' ,Trunc(12.75)); // Trunc(12.75) =  12
  ShowValue('Trunc(-12.75)= ',Trunc(-12.75));// Trunc(-12.75)= -12
  ShowValue('Int(12.75)= '   ,Int(12.75));   // Int(12.75)   =  12
  ShowValue('Int(-12.75)= '  ,Int(-12.75));  // Int(-12.75)  = -12
  ShowValue('Frac(12.75)= '  ,Frac(12.75));  // Frac(12.75)  = 0.75
  ShowValue('Frac(-12.75)= ' ,Frac(-12.75)); // Frac(-12.75) =-0.75
  ShowValue('Ceil(12.75)= '  ,Ceil(12.75));  // Ceil(12.75)  =  13
  ShowValue('Ceil(-12.75)= ' ,Ceil(-12.75)); // Ceil(-12.75) = -12
  ShowValue('Floor(12.75)= ' ,Floor(12.75)); // Floor(12.75) =  12
  ShowValue('Floor(-12.75)= ',Floor(-12.75));// Floor(-12.75)= -13
end;
```

See also: Mathematical methods

Sin

The sine of a number

Format

Function **Sin**(Value: real):real;

Description

The Sin function is a mathematical function giving the Sine value of a Number value given radians.
PI radians = 180 degrees

Example

```
Var
  f : real;
begin
  // The Sine of 30 degrees = 0.5
  f := Sin(PI/6);   // = 180/6 = 30 degrees
  ShowValue('Sin(PI/6) = ', f);
end;
```

See also: Mathematical methods

Sqrt

Gives the square root of a number

Format

Function **Sqrt**(Value: real):real;

Description

The Sqrt function returns the square root of a Number.
The square root of a negative value will generate a script error.

Example

```
Var
  R : real;
begin
  R := sqrt(25);
  ShowValue('result=',R); // result=5
end;
```

See also: Mathematical methods

Trunc

Returns the integer part of a floating point number

Format

Function Trunc(Value: real):int32;

Description

The Trunc function returns the integer part of a floating point number.
It returns this part as an Integer value.
Notes: The Int function does the same, but returns the integer in a floating point value.

Example

```
begin
  ShowValue('Round(12.75) = ',Round(12.75)); // Round(12.75) =  13
  ShowValue('Round(-12.75)= ',Round(-12.75));// Round(-12.75)= -13
  ShowValue('Trunc(12.75)= ' ,Trunc(12.75)); // Trunc(12.75) =  12
  ShowValue('Trunc(-12.75)= ',Trunc(-12.75));// Trunc(-12.75)= -12
  ShowValue('Int(12.75)= '   ,Int(12.75));   // Int(12.75)   =  12
  ShowValue('Int(-12.75)= '  ,Int(-12.75));  // Int(-12.75)  = -12
  ShowValue('Frac(12.75)= '  ,Frac(12.75));  // Frac(12.75)  = 0.75
  ShowValue('Frac(-12.75)= ' ,Frac(-12.75)); // Frac(-12.75) =-0.75
  ShowValue('Ceil(12.75)= '  ,Ceil(12.75));  // Ceil(12.75)  =  13
  ShowValue('Ceil(-12.75)= ' ,Ceil(-12.75)); // Ceil(-12.75) = -12
  ShowValue('Floor(12.75)= ' ,Floor(12.75)); // Floor(12.75) =  12
  ShowValue('Floor(-12.75)= ',Floor(-12.75));// Floor(-12.75)= -13
end;
```

See also: Mathematical methods

Messages / dialog functions

Methods

- DisplayMsg
- OpenDialog
- QueryStr
- QueryValue
- QueryYesNo
- SelectDir
- ShowMessage
- ShowSource
- ShowValue

DisplayMsg

Display a message with a title

Format

Procedure **DisplayMsg**(Title,Message:string);

Description

The **DisplayMsg** procedure displays a string of Text in a simple dialog with a title and two buttons:
- OK
- Cancel

If the user pushes the Ok button then the script will continue.
If the user pushes the Cancel button then It will be asked if he wants to stop the script or not. If he pushes on the Yes button then the script will stop.

It is used to inform the user of some information - no decision is needed by the user.
Insert carriage return character (#13) into the message string to generate multi line message

Examples

```
// Show a simple message
DispMessage('Important','Please turn off power supply');

// Show a multiple lines message
DispMessage('Important', 'Please check'+#13+'- Power supply'+#13+'- Communication
cable'+#13+'- Plugs');
```

See also: Messages / dialog funtions

OpenDialog

Select a file

Format

Function **OpenDialog**(InitialDir,filter:string):string;

Parameters

- InitialDir: Sets the starting directory in the dialog.
- Filter: This allows only certain file types to be displayed and selectable. The filter text is displayed in a drop down below the file name field. The following example selects for text files only:
  openDialog.Filter := 'Text files only|.txt';*
  The drop down dialog shows the description before the | separator. After the separator, you define a mask that selects the files you want.
  openDialog.Filter := 'Text and Word files only|.txt;*.doc';*
  above we have allowed two different file types, separated by a ;.
  openDialog.Filter := 'Text files|.txt|Word files|*.doc';*
  Above we have allowed text and Word files as two options in the drop down list.

Returned value

- Full file name with path and extension

Description

- This function is used to allow a user to select one file to open.

Example

```
Var
  FileName : string;
Begin
  FileName := OpenDialog('','*.ini');
  ShowMessage('Selected file is '+Filename);
End;
```

See also: Messages / dialog funtions

QueryStr

Ask the user to enter a string of character.

Format

Function **QueryStr**(Message:string): String;

Parameter

Message: The message can be separated into two parts by the character "|"
- The characters located before this special character are interpreted as the label of the QueryStr window.
- The characters located after this special character are interpreted as the default value to display
  If the message doesn't contain this special character, then the message is used only for the label on the left of the field.

Description

The **QueryStr** function displays a simple dialog box with the given Prompt message and the optional predefined string (separated by the character "|" in the message). It asks the user to enter a text in a text box of the dialog. Two buttons are located in the window: Ok and Cancel. The function will return when the user will push the Ok button. If the user cancels the dialog box, it will be asked to stop the script.

Example

```
// Put a Label component in a form, and in the event OnClick of this component,
// enter this script :
Begin
  Label1.Caption := QueryStr('Give a name for this recipe|'+ Label1.Caption);
End;
```

See also: Messages / dialog funtions

QueryValue

Ask to the user to enter a number.

Format

Function **QueryValue**(Message:string): real;

Parameter

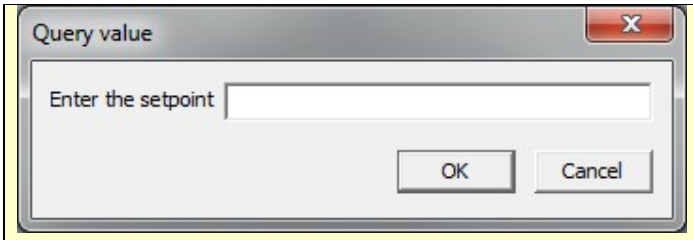
Message: The message can be separated into two parts by the character "|"
- The characters located before this special character are interpreted as the label of the QueryStr window.
- The characters located after this special character are interpreted as the default value to display
  If the message doesn't contain this special character, then the message is used only for the label on the left of the field.

Description

The **QueryValue** function displays a simple dialog box with the given Prompt message and the optional predefined string (separated by the character "|" in the message). It asks the user to enter a valid number in a text box on the dialog. Two buttons are located in the window: Ok and Cancel. The function will return only if the value is correct. If not, a message will be display and the user will have to enter a number again. If the user cancel the dialog box, it will be asked to stop the script.
Use to ask the user for simple data.

Example

```
R := QueryValue('Enter the setpoint');
```



See also: Messages / dialog funtions

QueryYesNo

Uses to ask the user to choose between Yes and No with a message

Format

Function **QueryYesNo**(Message:string): Boolean;

Returned value

- True (1) if the user presses on Yes.
- False (0) if the user presses on No.

Description

The **QueryYesNo** function displays a simple dialog box with the given Prompt message. Two buttons are located in the window: Yes and No. The function will return when the user will push a button.

See also: Messages / dialog funtions

SelectDir

Select a directory

Format

Function **SelectDir**(InitialDir:string):string;

Parameters

- InitialDir: Sets the starting directory in the dialog. The project folder will be selected if the string parameter is empty.

Returned value

- The selected directory always terminated by the "\" character.

Description

- This function is used to allow a user to select a directory.

Example

```
Var
  DirName : string;
Begin
  DirName := SelectDir('');
  ShowMessage('Selected directory is '+DirName);
End;
```

See also: Messages / dialog funtions

ShowMessage

Display a message

Format

Procedure **ShowMessage**(Message :string);

Description

The **ShowMessage** procedure displays a string of Text in a simple dialog with an OK button.
It is used to inform the user of some value - no decision is needed by the user.
Insert carriage return character (#13) into the string to generate multi line message display.

Examples

```
ShowMessage('Hello World'); // Show a simple message
ShowMessage('Hello'+#13+'World'); // Split this into two lines
```

See also: Messages / dialog funtions

ShowSource

Shows the source code window to see line in execution

Format

Procedure **ShowSource**(options:Integer);

Parameters

- Options: 0 (recommended) or 1

Description

The **ShowSource** procedure opens a window and displays the current line in execution.

The parameter *Option* can take two values:
- 0: Basic mode - only the source code is displayed with the current line in execution.
- 1: Advanced mode – internal data are also display, it is also possible to stop the script, to execute line by line or in continous.

Important: To use this function, the script must be compiled with the option “Add source info in byte code”. This option is available in the script editor menu: Options > Edit options
Recipes are always compiled with this option.

Example

```
Begin
  ShowSource(0);
End;
```

See also: Messages / dialog funtions

ShowValue

Display a value with a text

Format

Procedure **ShowValue**(Message : string; value : real);

Description

The **ShowValue** procedure displays a string of Text followed by a number in a simple dialog with an OK button.

It is used to inform the user of some value - no decision is needed by the user.
Insert carriage return character (#13) into the string to generate multi line message display.

Examples

```
ShowValue('Number of records=',Chamber1.Recorder1.RecordNum);
ShowValue('IMPORTANT'+#13+'Number of records=',Chamber1.Recorder1.RecordNum);
```

See also: Messages / dialog funtions

Modbus client module

The following methods are only available in the script of the Modbus driver.

Methods

- EnablePort: Enable, Disable the serial or socket module connected to the ASCII module.
- GetTagAddr: Get the tag address.
- GetTagFieldStr: Reads a tag property
- GetTagId: Returns the identifier of the tag specified by its name
- ReadTable: Reads data in the modbus table
- ReadTableFloat32: Reads a float in the modbus table
- ReadTableFromDevice: Reads data in the device
- ReadTagFromDevice: Reads a tag in the device
- ReadTagValue: Reads a tag in the table
- ReadTagValueEx: Reads a tag of a parameter of a tag
- SendTableToDevice: Sends data from the modbus table to the device
- SendTagToDevice: Sends a tag from the modbus table to the device
- WriteTable: Writes data in the modbus table
- WriteTableFloat32: Writes a float in the modbus table
- WriteTagValue: Sets a value to a tag identified by its id
- WriteTagValueEx: Sets a value to a tag identified by its name and channel number

EnablePort

(Modbus module)

Enable or disable the serial or socket module connected to the Modbus module.

Format

Function EnablePort(Status:int32;ShowLogMsg:boolean):int32

Parameters

- Status: Use to enable (1) or Close (0) or Destroy (-2) or return the status of the Port (-1)
- ShowLogMsg: true to log events messages when connecting/disconnecting

Returned value

- Status of the port:
  -1= No socket or serial module connected to the Modbus module.
  0 = Port (socket or serial module) is Closed or destroyed
  1 = Port (socket or serial module) is enabled

Description

This function is used to enable or disable (close or destroy) the port connected to the modbus module. It can also be used to know the status of the port.

- If the port is a serial port, disabling will close the com port and enabling will open it.
- If the port is a socket, closing will disconnect the socket from the server and enabling will reconnect the socket to the server. The destruction will destroy the memory socket (the socket will be re-created when it is activated)
- If no port is connected to the Modbus module then the function returns -1

Example

```
Begin
  EnablePort(0,false); // This will close or disconnect the port silently.
End;
```

See also: Modbus methods

GetTagAddr

(Modbus module)

Get the tag address

Format

Function **GetTagAddr**(TagName:String;Channel:word):integer;

Parameters

- TagName: String
- Channel: Channel number (starting at 1)

Returned value

The address of the tag

Description

Returns the address (register) of the tag identified by its name **TagName** and its **channel** number.
TagName is not case sensitive.
The **channel** number may be passed in parameter to the script.

Example

```
Var
  addr: integer;
  R : real;
Begin
  addr := GetTagAddr('SP',1); // SP is located at address 100
  WriteTable(Addr,130,2); // Write 130 in the modbus table at address "addr"
End;
```

See also: Modbus methods

GetTagFieldStr

(Modbus module)

Reads a tag property

Format

Function **GetTagFieldStr**(TagId,ColId:integer):string

Parameters

- TagId: Tag index (start at 0)
- ColId: Column number (see description, start at 1)

Returned value

The property of the tag

Description

This function reads a tag property identified by **TagId** and **channel** (start at 1) in the device, wait or not for reading and update the table.

If the tag does not exist, or if the ColId does not exist then an error message will be printed to the console.
GetTagFieldStr(0,1) will return the content of the column "Name" for the tag at index 0
GetTagFieldStr(0,2) will return the content of the column "R/W" for the tag at index 0

Columns list starting at 1:

Name	R/W	Address	Data type	Quantity	Frequency	ChCount	ChFirst	ChOffset	FxA	FxB	Link Type	Comment	Opt	Min	Max	Default	OnWrite	OnRead

Example

```
Begin
  ShowMessage( GetTagFieldStr(0,1) );
End;
```

See also: Modbus methods

GetTagId

(Modbus module)

Gets the tag index

Format

Function **GetTagId**(TagName:String;Channel:word):integer;

Parameters

- TagName : String
- Channel: Channel number (starting at 1)

Returned value

The index of the tag (start at 0)

Description

Returns the index of the tag identified by its name **TagName** and its **channel** number.
TagName is not case sensitive.
The **channel** number may be passed in parameter to the script.

Example

```
Var
  id: integer;
  R : real;
Begin
  if ReadTableFromDevice(20,100,2,true) // Read a register at address 100 and store at 20
    then ShowMessage('Yes, success');
    else ShowMessage('No failed');
  id := GetTagId('MV',1); // MV is located at address 100
  R  := ReadTagValue(id); // Get the value of MV in the modbus table
  ShowValue('Data =', R); // Display the value of MV
End;
```

See also: Modbus methods

ReadTable

(Modbus module)

Reads data in the modbus table

Format

Function **ReadTable**(Addr:word;Len:byte):Int64;

Parameters

- Addr: Address of the register (0 to 65535)
- Len : Number of byte to read (1 to 8)

Description

The modbus table is an array of 16 bits (word).

This function reads **len** bytes (up to 8 bytes) in the modbus table at address **Addr**.

Example

```
Begin
  WriteTableFloat(0,2.4); // Register 0 : $40 $19 $99 $9A
  ShowValue('Data 1=', ReadTable(0,1));     // Data 1=64 (=$40)
  ShowValue('Data 1 & 2=', ReadTable(0,2)); // Data 1 & 2= 16409 ($4019)
End;
```

See also: Modbus methods

ReadTableFloat32

(Modbus module)

Reads a float in the modbus table

Format

Function **ReadTableFloat32**(Addr:word):real;

Parameters

- Addr: Address of the register (0 to 65535)

Returned value

IEEE float (32 bits)

Description

The modbus table is an array of 16 bits (word).
This function reads a 32 bits float in the modbus table at address **Addr**

Example

```
Begin
  WriteTableFloat(0,2.4); // Register 0 : $40 $19 $99 $9A
  ShowValue('Data =', ReadTableFloat32(0));   // Data =2.40000009536743
End;
```

See also: Modbus methods

ReadTableFromDevice

(Modbus module)

Reads data in the device

Format

Function **ReadTableFromDevice**(SrcAddr,DestAddr,len:Word;WaitForReading:boolean):boolean;

Parameters

- SrcAddr: Address to read in the device.
- DestAddr: Address where to store the data in the modbus table
- Len: number of bytes to read
- WaitForReading: not used, always considered as true

Returned value

False after a time out or true if success.

Description

Reads **len** bytes at address **SrcAddr** in the device and store data at address **DestAddr** in the modbus table.
The call to this function always result a request to the device.
The script will continue only after receiving the frame from the device or after a time out.

The time out value is calculated following this formula:

Time out = (NbComNOK+2)*ComTMO

With
NbComNOK: this value is defined in the modbus regulator as “Nb com NOK”
ComTMO: this value is defined in the modbus regulator as “Time out (ms)”

Example

```
Var
  id: integer;
  R : real;
Begin
  if ReadTableFromDevice(20,100,2,true) // Read a register at address 100 and store at 20
    then ShowMessage('Yes, success');
    else ShowMessage('No failed');
  id := GetTagId('MV',1); // MV is located at address 100
  R  := ReadTagValue(id); // Get the value of MV in the modbus table
  ShowValue('Data =', R); // Display the value of MV
End;
```

See also: Modbus methods

ReadTagFromDevice

(Modbus module)

Reads a tag in the device

Format

Function **ReadTagFromDevice**(TagName:string; Channel:word; ReadingMode:byte):integer;

Parameters

- TagName: Name of the tag to send
- Channel: Channel number (start at 1)
- ReadingMode:
  0: A reading request is initiated but the script does not wait to receive the answer, the tag will be updated later, after receiving the answer.
  1: A reading request is initiated and the script waits until receiving the answer.
  2: The script does not trig a reading.

Returned value

- The tag index or -1 if the tag does not exist.

Description

This function reads a tag identified by **TagName** (not case sensitive) and **channel** (start at 1) in the device, wait or not for reading and update the table.

Returns the index of the tag or -1 if the tag doesn’t exist (this may happen also if the channel is not connected to a sub equipment)
If the tag does not exist, an error message will be printed to the console.

Example

```
Begin
  ReadTagFromDevice('SP',1,1);
End;
```

See also: Modbus methods

ReadTagValue

(Modbus module)

Reads a tag in the table

Format

Function **ReadTagValue**(TagId:word):real;

Parameters

- TagId: Tag index (number of tag as it appears in the device analyser, start at 0)

Returned value

Real value

Description

Reads a tag of any type in the modbus table identified by TagId.
The linearization (using FxA and FxB) is applied before returning the value.
The call to this function does not result a request to the device.
if the tag identified by IdTag does not exist, an error message will be printed to the console.

Example

```
Var
  id: integer;
  R : real;
Begin
  WriteTable(100,23,4);
  id := GetTagId('MV',1);
  R  := ReadTagValue(id);
  ShowValue('MV=', R);
End;
```

See also: Modbus methods

ReadTagValueEx

(Modbus module)

Reads a tag of a parameter of a tag

Format

Function **ReadTagValueEx**(TagName:string;Channel:integer):real;

Parameters

- TagName: Name of the tag or Tag parameter
- Channel: Channel number

Description

Reads a tag of any type in the modbus table identified by the tagname
The linearization (using FxA and FxB) is applied before returning the value.
The call to this function does not result a request to the device.

If the tag identified by the Tagname does not exist, an error message will be printed to the console.

The TagName can also be followed by:
.FxA
.FxB
.Min
.Max
.Frequency
Exemple of TagName: TSP.FxA will return the value of FxA

Example

```
Var
  R : real;
Begin
  WriteTable(100,23,4);
  R  := ReadTagValueEx('MV',1);
```

```
  ShowValue('MV=', R);
  R  := ReadTagValueEx('MV.FxA',1);
  ShowValue('MV.FxA=', R);

End;
```

See also: Modbus methods

SendTableToDevice

(Modbus module)

Sends data from the modbus table to the device

Format

Function **SendTableToDevice**(SrcAddr,DestAddr,Len:Word):boolean;

Parameters

- SrcAddr: Address in the modbus table (data source)
- DestAddr: Address in the device (Destination address)
- Len: Number of bytes to write

Returned value

False after a time out or true if success.

Description

This function sends **Len** bytes from the internal modbus table at address **SrcAddr** to the device at address **DesAddr**.

Example

```
Begin
  WriteTableToDevice(0,100, 2); // send the register at 0 to address 100 into the device
End;
```

See also: Modbus methods

SendTagToDevice

(Modbus module)

Sends a tag from the modbus table to the device

Format

Function **SendTagToDevice**(TagName:string ; Channel:word ; WaitForWriting:boolean):integer;

Parameters

- TagName: Name of the tag to send
- Channel: Channel number
- WaitForWriting: True to wait for sending the tag to the device

Returned value

The tag index or -1 if the tag does not exist.

Description

This function sends a tag identified by **TagName** (not case sensitive) and **channel** (start at 1) to the device, wait or not for writing.

Returns the Identifier of the tag or -1 if the tag doesn't exist (this may happen also if the channel is not connected to a sub equipment)

If the tag does not exist, an error message will be printed to the console.

Example
```
Begin
  SendTagToDevice('SP',1,true);
End;
```

See also: Modbus methods

WriteTable
(Modbus module)

Writes data in the modbus table

Format

Procedure **WriteTable**(Addr:word;Value:Int64;Len:byte);

Parameters

- Addr: Address of the register (0 to 65535)
- Value: value to write
- Len: number of byte to write (1 to 8)

Description

Writes up to 8 bytes in the modbus table (this is not sent to the device)
Due to the order of the modbus table, the first byte of a register is the most significant byte of the register.

Example
```
Begin
  WriteTable(0, 1, 2); // Register 0 contains 1 ($00 $01)
  WriteTable(0, 1, 1); // Register 0 contains >=256: write 1 to the MSB ($01 $aa)
  WriteTable(0,256,2); // Register 0 contains 256 ($01 $00)
End;
```

See also: Modbus methods

WriteTableFloat32
(Modbus module)

Writes a float in the modbus table

Format

Procedure **WriteTableFloat32**(Addr:word;Value:Real)

Parameters

- Addr: Address of the register (0 to 65535)
- Value: value to write

Description

Writes a float 32 bits (IEEE) in the modbus table (this is not sent to the device)

Example
```
Begin
  WriteTableFloat(0,2);   // Register 0 : $40 $00 $00 $00
  WriteTableFloat(0,2.4); // Register 0 : $40 $19 $99 $9A
End;
```

See also: Modbus methods

WriteTagValue

(Modbus module)

Sets a value to a tag identified by its id

Format

Procedure **WriteTagValue**(IdTag:word;Value:real;SendToDevice:Byte);

Parameters

- IdTag: The index of the tag (see GetTagId) (start at 0)
- Value: Value to set to the tag (the linearization will be applied)
- SendToDevice: 0 or false =don't send it to the device, 1 or true=write the tag to the device, 2=write the tag to the device and wait for the value to be sent.

Description

This procedure stores the **value** identified by **IdTag** in the modbus table and, if required, send it to the device (only if parameter SendToDevice is different of zero).
The reversed linearization using FxA and FxB is applied before to store the value into the table.

To be send to the device, the tag must be of type R/W or Write.

If the tag identified by IdTag does not exist, an error message will be printed to the console.
If the tag cannot be sent to the device, the function will return after the timeout defined in the modbus module.

Example

```
Var
  id: integer;
Begin
  id := GetTagId('MV',1);
  WriteTagValue(Id, 123, true);
End;
```

See also: Modbus methods

WriteTagValueEx

(Modbus module)

Sets a value to a tag identified by its name and channel number

Format

Procedure **WriteTagValueEx**(TagName:string;Channel:integer;Value:real;SendToDevice:Boolean);

Parameters

- TagName: Name of the tag or Tag parameter
- Channel: Channel number
- Value: Value to set to the tag (the linearization will be applied)
- SendToDevice: 0=don't send it to the device, 1=write the tag to the device, 2=write the tag to the device and wait for the value to be sent.

Description

This procedure writes a **value** to the tag identified by **TagName** and **Channel** and send it to the device if required.
The reversed linearization using FxA and FxB is applied before to store the value into the table.
If the parameter SendToDevice is true, then the tag will be sent to the device.

The TagName can also be followed by:
.FxA
.FxB
.Min
.Max

.Frequency
Exemple of TagName: TSP.FxA will change the value of FxA

To be send to the device, the tag must be of type R/W or Write.

If the tag does not exist, an error message will be printed to the console.
If the tag cannot be sent to the device, the function will return after the timeout defined in the modbus module.

Example
```
Begin
  WriteTagValueEx('SP',1, 123, true); // Write 123 to the tag SP and send it to the device
  WriteTagValueEx('MV.FxA',1, 1.23, false);  // Change the FxA factor
End;
```

See also: Modbus methods

OLE

OLE Automation: Object Linking and embedding

Methods
- App.OLE.Close
- App.OLE.ExportPDF
- App.OLE.LoadDocument
- App.OLE.Poke
- App.OLE.Request
- App.OLE.RunMacro
- App.OLE.SaveDocument
- App.OLE.ShowDocument
- App.OLE.TableAddRows
- App.OLE.TableCreate
- App.OLE.TableSetCell

App.OLE.Close

Closes the document previously opened by App.OLE.LoadDocument

Format

Function **App.OLE.Close**(DocId:integer):boolean;

Parameter

- DocId: a number identifying the document (0 to 10)

Returned value

- True is success

Description

This function closes the document identified by Doc Id opened by the function App.OLE.LoadDocument.

Example
```
Begin
  App.OLE.LoadDocument(1,TDT_Writer,'c:\temp\MyTemplate.odt');
  App.OLE.Poke(1,'','Name','Joe Garden');
  App.OLE.Poke(1,'','RecipeName','GaAs_Recipe12.rcp');
  App.OLE.ExportPDF(1,'c:\temp\test.pdf');
  App.OLE.Close(1);
End;
```

See also: OLE functions

App.OLE.ExportPDF

Exports the document opened by App.OLE.LoadDocument to PDF File

Format

Function **App.OLE.ExportPDF**(DocId:integer;OutputFileName:String):boolean;

Parameter

- DocId: a number identifying the document (1 to 10) see App.OLE.LoadDocument
- OutputFileName: Filename with path and extension.

Returned value

- True is success

Description

This function exports the document identified by *DocId* to a PDF file.

Example

```
Begin
  App.OLE.LoadDocument(1,TDT_Writer,'c:\temp\MyTemplate.odt');
  App.OLE.Poke(1,'','Name','Joe Garden');
  App.OLE.Poke(1,'','RecipeName','GaAs_Recipe12.rcp');
  App.OLE.ExportPDF(1,'c:\temp\test.pdf');
  App.OLE.Dispose(1);
End;
```

See also: OLE functions

App.OLE.LoadDocument

Opens a text document or a spreadsheet document

Format

Function **App.OLE.LoadDocument**(DocId:integer; FileName:String):integer;

Parameter

- DocId: a number identifying the document (0 to 10)
- DocType: See bellow the constant beginning by TDT_xxx
- FileName: Filename with path and extension of the document to load.

Returned value

One of the following values
- **(-1) = ERROR**
- **0 = Unknown**
- **1 = Writer**
- **2 = Calc**
- **3 = Draw**
- **4 = Math**
- **5 = Word**
- **6 = Excel**

Description

Doctype must be one of the following contant defined in the stfconst unit:
TDT_Unknown = 0
TDT_Writer = 1
TDT_Calc = 2
TDT_Word = 3

TDT_Excel = 4
TDT_Draw = 5
TDT_Math = 6
TDT_Impress = 7

This function opens a file and identify it by DocId. This identifier will be used by others OLE functions as the document identifier.

Example

```
Uses stdconst;
Begin
  App.OLE.LoadDocument(1,TDT_Writer,'c:\temp\MyTemplate.odt');
  App.OLE.Poke(1,'','Name','Joe Garden');
  App.OLE.Poke(1,'','RecipeName','GaAs_Recipe12.rcp');
  App.OLE.ExportPDF(1,'c:\temp\test.pdf');
  App.OLE.Dispose(1);
End;
```

See also: OLE functions

App.OLE.Poke

Writes a string of characters in a document or spreadsheet.

Format

Function **App.OLE.Poke**(DocId:integer;Sheet,Bookmark,SValue:String):boolean;

Parameter

- **DocId**: a number identifying the document (0 to 10) see App.OLE.LoadDocument
- **Sheet** : Sheet name or Sheet number (#xx) if the document is a spreadsheet or empty if the document is a text document.
- **Bookmark**: Bookmark of the document or spread sheet.
  If the document is a spreadsheet it can represent the cell reference (RxCy) or something like 'A3'
- **SValue**: String to write.

Returned value

- True is success

Description

This function writes a string of characters in a document or spreadsheet at the location identified by *Bookmark*.

Example

```
Begin
  App.OLE.LoadDocument(1,TDT_Writer,'c:\temp\MyTemplate.odt');
  App.OLE.Poke(1,'','Name','Joe Garden');
  App.OLE.Poke(1,'','RecipeName','GaAs_Recipe12.rcp');
  App.OLE.ExportPDF(1,'c:\temp\test.pdf');
  App.OLE.Dispose(1);
End;
```

See also: OLE functions

App.OLE.Request

Reads data from a document or spreadsheet.

Format

Function **App.OLE.Request**(DocId:integer;Sheet,Bookmark,Default:String):boolean;

Parameter

- **DocId**: a number identifying the document (0 to 10) see App.OLE.LoadDocument

- **Sheet** : Sheet name or Sheet number (#xx) if the document is a spreadsheet or empty if the document is a text document.
- **Bookmark**: Bookmark of the text document or spreadsheet document.
  If the document is a spreadsheet it can represent the cell reference (RxCy) or something like 'A3'
- **Default**: Default value to return in case of error.

Returned value

- The data requested or the default value if an error occured

Description

This function reads data to a spread sheet or a text document identified by *DocId*, the sheet name *Sheet* or the sheet number if the document is a spreadsheet and the *Bookmark*. The bookmark can represent also the cell reference of a spreadsheet.

Example

```
Begin
  App.OLE.LoadDocument(1, ,TDT_Calc, 'C:\DDE\calcTemplate.ods');
  ShowMessage(App.OLE.Request(1,'','R2C2','Error !!!'));
  ShowMessage(App.OLE.Request(1,'','A1','Error !!!'));
  ShowMessage(App.OLE.Request(1,'','MyBookmark','Error !!!'));
  ShowMessage(App.OLE.Request(1,'MySheet2','MyBookmark','Error !!!'));
  App.OLE.Dispose(1);
End;
```

See also: OLE functions

App.OLE.RunMacro

Executes a macro in a document or spreadsheet

Format

Function **App.OLE.RunMaro**(DocId:integer;MacroName,params:string):boolean;

Parameter

- **DocId**: a number identifying the document (0 to 10) see App.OLE.LoadDocument
- **MacroName** must be followed by '?language=<*languageType*>&location=<*application or document*>'
- **Params** is reserved for future used.

Returned value

- True is success

Description

This function executes the macro *MacroName* in the document identified by *DocId*.
The MacroName must be followed by a text specifying le language type and the location.

- The language type can be **Basic**, **Beanshell**, **JavaScript** or **Python**
- The location can be **application** or **document**

Example

```
Begin
  App.OLE.LoadDocument(1,TDT_Calc,'C:\DDE\calcTemplate.ods');
  App.OLE.RunMacro(1,'MyTestMacro?language=Basic&location=document','');
  App.OLE.Dispose(1);
End;
```

See also: OLE functions

App.OLE.SaveDocument

Saves the document as, with a new name and new type

Format

Function **App.OLE.SaveDocument**(DocId:integer,FName,Format:String):integer;

Parameter

- **DocId**: a number identifying the document (0 to 10) see App.OLE.LoadDocument
- **FName:** File name with path an extension.
- **Format:** Depending on the type of the application, see the library OLEConst.pas for more details.

  Example of Format for libre office:
  oFormat_odt = 'writer8';
  oFormat_html = 'writerweb8_writer';
  oFormat_ods = 'calc8';
  oFormat_odp = 'impress8';
  oFormat_xlsx = 'Calc MS Excel 2007 XML'; // Same constant uses for the files .xlsx or slsm
  oFormat_docx = 'MS Word 2007 XML'; // Same constant uses for the files .docx or docm
  oFormat_doc = 'MS Word 97';
  oFormat_ppt = 'MS PowerPoint 97'; // Same as pps
  oFormat_pdf_calc='calc_pdf_Export';
  oFormat_pdf_writer='writer_pdf_Export';
  oFormat_pdf_impress='impress_pdf_Export';

Returned value

- Returns 1 if success.

Description

Saves the current document in a new file and possibly in a new format. For example, if the current document loaded by LoadDocument is a libre office writer file of type .odt, it can be saved in a MS Word file format with the extension .DOC

Example

```
// In this example, we load an excel document with Libre Office Calc, fill it and save it in Excel format
Uses OLEConst;
Const
  DocId = 1;
Var
  FName : String;
Begin
  FName := App.ProjectDirectory+'Template\Report\QCS.xlsx';
  if App.OLE.LoadDocument(DocId,TDT_calc,FName)<0 then// 2=TDT_Calc, we use libre office Calc to open the
file
  Begin
    ShowMessage('Cannot load the file '+FName);
    Exit;
  end;
  // Fill the Date and time
  App.OLE.Poke(DocId,'','DateTime',GetDateTime('dd/mm/yy hh:nn:ss'));
  // Fill the recipe name
  App.OLE.Poke(DocId,'','RecipeName',Recipe.RecipeName);
  FName := App.ProjectDirectory+'Record\'+FormatDateTime('yyyy-mm-dd_hh-mm-ss',GetNow)+'_Report.xlsx';
  App.OLE.SaveDocument(DocId,FName, ,oFormat_xlsx); // See library OLEConst.pas for all formats
End;
```

See also: OLE functions

App.OLE.ShowDocument

Shows the document identifed by DocId as normal, maximized, minimized etc..

Format

Procedure **App.OLE.ShowDocument**(DocId,nCmdShow : integer);

Parameter

- **DocId**: a number identifying the document (0 to 10) see App.OLE.LoadDocument

Returned value

- Returns 1 if success.

Description

Controls how the window is to be shown. See the Microsoft help center about ShowWindow function for all the details.
Here is the list of all nCmdShow parameter values:
SW_HIDE = 0;
SW_SHOWNORMAL = 1;
SW_NORMAL = 1;
SW_SHOWMINIMIZED = 2;
SW_SHOWMAXIMIZED = 3;
SW_MAXIMIZE = 3;
SW_SHOWNOACTIVATE = 4;
SW_SHOW = 5;
SW_MINIMIZE = 6;
SW_SHOWMINNOACTIVE = 7;
SW_SHOWNA = 8;
SW_RESTORE = 9;
SW_SHOWDEFAULT = 10;
SW_FORCEMINIMIZE = 11;

Example

```
Uses OLEConst;
Const
  DocId = 1;
Var
  FName : String;
Begin
  FName := App.ProjectDirectory+'Template\Report\QCS.xlsx';
  if App.OLE.LoadDocument(DocId,2,FName)<0 then// 2=TDT_Calc, we use libre office Calc to open the file
  Begin
    ShowMessage('Cannot load the file '+FName);
    Exit;
  end;
  App.OLE.ShowDocument(DocId,SW_MAXIMIZE); // Show the loaded document in full screen
End;
```

See also: OLE functions

App.OLE.TableAddRows

Adds rows to a table in a Writer document or Word document.

Format

Procedure **App.OLE.TableAddRows**(DocId, TableId, RowCount: integer);

Parameter

- **DocId**: a number identifying the document (0 to 10) see App.OLE.LoadDocument
- **TableId:** Identify the table in the document. The first table is 1, the second one is 2 etc..
- **RowCount:** Number of rows to add to the table.

Description

Adds rows to an existing table in a Libre Office Writer document or MSWord document.

Example

```
Uses OLEConst;
Const
  DocId = 1;
  Row = 1;
Var
  FName : String;
Begin
```

```
  FName := App.ProjectDirectory+'Template\Report\QCS.xlsx';
  if App.OLE.LoadDocument(DocId,2,FName)<0 then// 2=TDT_Calc, we use libre office Calc to open the file
  Begin
    ShowMessage('Cannot load the file '+FName);
    Exit;
  end;
  App.OLE.TableAddRows(DocId,1,1); // Add a new row to the first table of the document
  App.OLE.TableSetCell(DocId,1,1,2,'R1C2'); // Fill the cell R1C2 of the first table
End;
```

See also: OLE functions

App.OLE.TableCreate

Creates a table in a Writer document or Word document.

Format

Function **App.OLE.CreateTable**(DocId: Integer; Bookmark: String; RowCount, ColCount, TableColor: integer):integer;

Parameter

- **DocId**: a number identifying the document (0 to 10) see App.OLE.LoadDocument
- **BookMark**: location of the table to create in the document;
- **RowCount:** Number of rows to add to the table.
- **ColCount:** Number of rows to add to the table.
- **TableColor:** Background color of the table
  Example: $FF0000 is Blue, $FF00 is Green, $FF is red; $C0C0C0 is silver, $FFFFFF is white. Refer to the constant colors list defined in the library file StdConst.pas for more details.

Returned value

- Returns 1 (if Libre Office Writer document) or the number of tables present in the document (if MSWord) or zero if error.

Description

Creates a table in a Libre Office Writer document or MSWord document identified by DocId, at the location of the Bookmark with *RowCount* rows, *ColCount* columns and *TableColor* background color.

Example

```
Uses OLEConst;
Const
  DocId = 1;
  Row = 1;
Var
  FName : String;
Begin
  FName := App.ProjectDirectory+'Template\Report\QCS.xlsx';
  if App.OLE.LoadDocument(DocId,2,FName)<0 then// 2=TDT_Calc, we use libre office Calc to open the file
  Begin
    ShowMessage('Cannot load the file '+FName);
    Exit;
  end;
  App.OLE.TableCreate(DocId,'MyTableBookmark',1,6,$FF0000); // Create a new table in the document DocId
  App.OLE.TableAddRows(DocId,1,1); // Add a new row to the first table of the document
  App.OLE.TableSetCell(DocId,1,1,2,'R1C2'); // Fill the cell R1C2 of the first table
End;
```

See also: OLE functions

App.OLE.TableSetCell

Fills a cell of a table.

Format

Procedure **App.OLE.TableSetCell**(DocId, TableId, Rx , Cx : integer; Content : String):integer;

Parameter

- **DocId**: a number identifying the document (0 to 10) see App.OLE.LoadDocument
- **TableId:** Number of the table in the document beginning at 1.
- **Rx:** Row number
- **Cx:** Column number
- **Content:** String character to write in the cell

Description

Fills the cell at coordinates *Rx,Cx* in the table *TableId* of a Libre Office Writer document or an MSWord document identified by *DocId* with the string of characters *Content.*

Example

```
Uses OLEConst;
Const
  DocId = 1;
Var
  FName : String;
Begin
  FName := App.ProjectDirectory+'Template\Report\QCS.xlsx';
  if App.OLE.LoadDocument(DocId,2,FName)<0 then// 2=TDT_Calc, we use libre office Calc to open the file
  Begin
    ShowMessage('Cannot load the file '+FName);
    Exit;
  end;
  App.OLE.TableCreate(DocId,'MyTableBookmark',1,6,$FF0000); // Create a new table in the document DocId
  App.OLE.TableAddRows(DocId,1,1); // Add a new row to the first table of the document
  App.OLE.TableSetCell(DocId,1,1,2,'R1C2'); // Fill the cell R1C2 of the first table
End;
```

See also: OLE functions

Profile

Methods

- GetProfileName
- GetProfileValue

GetProfileName

Returns the name of a profile.

Format

Function **GetProfileName**(NumProfile:integer):String;

Parameters

- NumProfile: Number of profile beginning at 1

Returned value

- The name of the profile identified by *NumProfile*

Description

The GetProfileName returns the name of the profile identified by the parameter NumProfile.
If no profile corresponds to the NumProfile parameter then an empty string is returned.

Example

```
Var
  i : integer;
  S : string;
Begin
  i := 1;
  Repeat
```

```
    S := GetProfileName(i);
    If length(S)>0 then writeconsole(S);
    i := i+1;
  Until length(S)=0;
end;
```

See also: Profile methods

GetProfileValue

Returns the setpoint value of a profile

Format

Function **GetProfileValue**(NumProfile:integer; Timer,slope,StartMV,TargetSP:real):Real;

Parameters

- NumProfile: Number of profile beginning at 1
- Timer: elapsed time since the start of the ramp in second
- Slope: slope of the ramp in unit/second
- StartMV: Start value in the unit
- TargetSP: value to reach in the unit

Returned value

- Setpoint to send to the device

Description

The GetProfileValue uses the profile defined by NumProfile and other parameters to calculate the corresponding value.

Example

```
Var
  SP,MaxTime : Real;
Begin
  ResetTimer(1);
  MaxTime := abs( (StartMV-RampSP)/Slope);
  Repeat
  // ...
    SP := GetProfileValue(1, GetTimer(1), Slope, StartMV, RampSP);
  // ...
  Until (GetTimer(1) > MaxTime);
end;
```

See also: Profile methods

Properties

Methods

- GetPropStr
- GetPropType
- GetPropValue
- SetPropStr
- SetPropValue

SetPropValue

Assigns the value of a property identified by its name given as a string

Format

Procedure **SetPropValue**(PropName:String; Value:real);

Parameters

- PropName: Name of the property to set

- Value: Value to set

Description

The **SetPropValue** procedure is used to change a property by using its name in string format.
The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.
If the property does not exist, a script error will be displayed and logged and the script will stop.
Uses GetPropType before to check is the property exists.

Example

```
begin
  SetPropValue('App.Tag',12);
  ShowValue('App.Tag=',App.Tag); // App.Tag=12
end;
```

See also: Properties

GetPropValue

Returns the value of a property identified by its name as a string

Format

Function **GetPropValue**(PropName:String;default:real) : real;

Parameters

- PropName: Name of the property to read
- Default: Default Value to return

Description

The GetPropValue function is used to read a property value by using its name in string format.
The intrinsic value of this function is to allow reading of component properties without the need to hard code the property name. This allows mass visual component processing at run time.
If the property does not exist, the function will return the default value.

Example

```
begin
  App.Tag := 34;
  ShowValue('App.Tag=',GetPropValue('App.Tag',-1));
end;
```

See also: Properties

SetPropStr

Assigns a string property identified by its name given as a string

Format

Procedure **SetPropStr**(PropName, ValueStr:String);

Parameters

- PropName: Name of the property to read
- ValueStr: String to set

Description

The **SetPropStr** procedure is used to change a property of type string by using its name in string format.
The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.
If the property does not exist, a script error will be displayed and logged and the script will stop.
Uses GetPropType before to check is the property exists.

Example

```
begin
  SetPropStr('App.Name','Hello');
  ShowMessage('App. Name ='+App.Name); // App.Name='Hello'
end;
```

See also: Properties

GetPropStr

Reads a string property from is name given as a string

Format

Function **GetPropStr**(PropName, default:string):string;

Parameters

- PropName: Name of the property to read
- Default: Default String to return

Description

The GetPropStr function is used to read a property of type String by using its name in string format.
The intrinsic value of this function is to allow reading of component properties without the need to hard code the property name. This allows mass visual component processing at run time.
If the property does not exist, the function will return the default string.

Example

```
Begin
  App.Name := 'Hello';
  ShowMessage('App.Name='+GetPropStr('App.Name',''));
end;
```

See also: Properties

GetPropType

Returns the type of a property

Format

Function **GetPropType**(PropName:String):Integer;

Parameters

- PropName: Name of the property to read the type

Returned value

- 0 : Unknown property,
- 1 : byte
- 2 : Boolean
- 3 : word
- 4 : uint32
- 5 : Int16
- 6 : int32
- 7 : int64
- 8 : real
- 9 : float32
- 10: uint64
- 128 : String (252 characters max)
- >128: String of length x-128

Description

The GetPropType function is used to read the type of a property by using its name in string format.

If the property does not exist, the function will return zero.

Example

```
begin
  ShowValue('Type of App.Name=',GetPropType('App.Name')); // Type of App.Name=128
end;
```

See also: Properties

GetHandleStr

Converts the handle of a script into a string

Format

Function **GetHandleStr**(Handle:integer; bitmap:byte):string;

Parameters

- Handle: Property passed to the script
- bitmap: define which data to convert

Returned value

- String property

Description

The GetHandleStr function is used to convert the handle of a script into a string.
Script of events which are used in the equipments, sub equipments and regulators have a parameter **handle**.

Some bits of **bitmap** identified which data to convert:
- b0 (1) = regulator
- b1 (2) = sub equipment
- b2 (4) = Equipment
- b3 (8) = Chamber
- b4-b7 = not used

Each bytes of the **handle** is the index of regulator (LSB), sub equipment, equipment and chamber (MSB).

The returned string is the concatenation of name of each element separate by a “.”

Example: GetHandleStr could returns: “Chamber1.Manipulator.Shutter” if bitmap is 7

Example

```
// Example of script which is executed in a modbus driver
Var
  NumChmbr : Integer;
Begin
  NumChmbr := Shr(Handle,24)+1; // Get the chamber number
  LogEvent(NumChmbr,800,3,0,GetHandleStr(Handle,1)+' - Driver Initialized');
end;
```

Recipe

Methods
- AddRampToList
- Layer
- LoadFromFile
- Log
- Pause
- Resume
- Skip
- Start
- Stop

AddRampToList

Add a ramp to the list to display at run time and stop at the end.

Format

Procedure **AddRampToList**(TagName:String);

Parameters

- TagName is the tag name of the ramp setpoint (RampSP or RampOP).

Description

This function adds a ramp to the list of ramps that are executed in the recipe.
The list of ramps is visible during the execution of the recipe in the "Ramps" tab of the recipe inspector.
At the end of the recipe, the ramps which remain in execution are displayed in a window and stopped automatically if no user action is taken.

Example

```
Begin
  AddRampToList('effusioncell1.temperature.RampSP');
  AddRampToList('effusioncell2.temperature.RampOP');
End;
```

See also: Recipe functions

Layer

Processes a layer

Format

Procedure **Layer**(Comment:string;TimeMS:integer);

Parameters

- Comment: Text line to be log in the recipe inspector
- TimeMS: time of the layer in millisecond

Description

This function can be called only in a recipe.
When executing this function, the comment will be logged in the history list of the recipe inspector and the script will wait for the given time.

It is important to understand that the time is the cumulative time calculated from the start of the recipe.

Example:
If the recipe waits for a temperature reaches a value.
If the time to reach this value is 1 minute and the next layers duration is less than one minute, then the next layers will be executed in the same time as soon as the temperature will be reached. In the following example the layers T1, T2 and T3 will be executed at the same time.

```
Chamber1.Cell1.Shutter:= open;
Repeat
Until (Chamber1.Cell1.Temperature.MV>700);
Chamber1.Cell1.Shutter:= close;
Layer('layer T1', 30000);
Chamber1.Cell2.Shutter:= open;
Layer('layer T2', 10000);
Chamber1.Cell2.Shutter:= close;
Layer('layer T3', 10000);
```

To avoid that behaviour, uses the layer time zero after a waiting instruction:

```
Chamber1.Cell1.Shutter := open;
Repeat
Until (Chamber1.Cell1.Temperature.MV>700);
Layer('Reset time',0);
Chamber1.Cell1.Shutter:= close;
Layer('layer T1', 30000);
Chamber1.Cell2.Shutter := open;
Layer('layer T2', 10000);
Chamber1.Cell2.Shutter := close;
Layer('layer T3', 10000);
```

Example

```
Begin
  As1_VAC500.shutter.Control := On;
  Layer('My first layer',30000);
  As1_VAC500.shutter.Control := off;
  Layer('My second layer',30000);
End;
```

See also: Recipe functions

LoadFromFile

Loads a recipe

Format

Function **LoadFromFile**(Filename:string;Priority:byte):boolean;

Parameters

- Filename: File name of the recipe (.pas or .rcp)
- Priority: 0 (lowest) to 5 (highest)

Returned value

- Returns true if the recipe is loaded successfully.

Description

This function must be called from another script.
The full path of the recipe must be specified.

The filename can be:
- The full path with drive letter (ex: C:\riber\CRYSTAL XE_Project\Recipe\C21_III_V\test.pas)
- Relative path to the project (ex: \C21_III_V\test.pas)

Bit organization of Bitmap_options:
- b0 = execute pre recipe
- b1 = execute post recipe

Values of Bitmap_options
- 0 = no other recipe is executed.
- 1 = the pre-recipe is executed.
- 2 = the post-recipe is executed.
- 3 = the pre-recipe and the post-recipe are executed.

Pre-recipe and post-recipe file names are defined in the hardware configuration.

Example

```
Begin
  if Chamber1.Cell1.Temperature.MV>700 then
  Begin
    Chamber1.Recipe.LoadFromFile('test.pas',0);
    Chamber1.Recipe.Start(0);
  End;
End;
```

See also: Recipe functions

Log

Adds a new line in the recipe log

Format

Procedure **Log**(Horodate:Boolean; Message: String);

Description

Stores a text line in the history list of the recipe.
This function is available only if a recipe is loaded.
This function must be called with the prefix “Recipe” (Recipe.log) to avoid confusion with the logarithmic function.

Example

```
Begin
  Recipe.log(true,'The gate is opened');
End;
```

See also: Recipe functions

Pause

Suspends the execution of the recipe

Format

Procedure **Pause**;

Description

This function suspends the current recipe.
This function can be also called from another script.
In that case, the full path of the recipe must be specified.

Example

```
Begin
  (...)
  if Chamber1.Cell1.Temperature.MV<700 then Pause;
  (...)
End;
```

See also: Recipe functions

Resume

Resumes the execution of the recipe after a pause

Format

Procedure **Resume**;

Description

This function resumes the current recipe after a pause.
This function can be also called from another script.
In that case, the full path of the recipe must be specified.

Example

```
Begin
  (...)
  if Chamber1.Cell1.Temperature.MV>700 then Chamber1.Recipe.Resume;
  (...)
End;
```

See also: Recipe functions

Start

Starts a recipe

Format

Function **Start**(bitmap_options:word):boolean;

Parameters

- bitmap_options: define if the recipe must execute the pre recipe and the post recipe.

Returned value

- Returns true if the recipe is started.

Description

This function must be called from another script.
A recipe must be loaded, see the function LoadFromFile for more information.

Bit organization of Bitmap_options:
- b0 = execute pre recipe
- b1 = execute post recipe

Pre recipe and post recipe file name are defined in the hardware configuration.

Example
```
Begin
  if Chamber1.Cell1.Temperature.MV>700 then Chamber1.Recipe.Start(0);
End;
```

See also: Recipe functions

Stop

Stops the recipe

Format

Procedure **Stop**;

Description

This function stops the current recipe.
This function can be also called from another script.
In that case, the full path of the recipe must be specified.

Example
```
Begin
  (...)
  if Chamber1.Cell1.Temperature.MV<700 then Stop;
  (...)
End;
```

See also: Recipe functions

Skip

Skips a layer of a recipe

Format

Procedure **Skip**;

Description

This function must be called from another script.
The full path of the recipe must be specified.

Example

```
Begin
  if Chamber1.Cell1.Temperature.MV>700 then Chamber1.Recipe.Skip;
End;
```

See also: Recipe functions

Recorder

Methods

- Start
- Stop

Recorder1.Start

Starts the recorder

Format

Function **Recorder1.start** : Boolean;

Returned value

- This function returns true is the recorder is started.

Description

A template must be loaded (default or custom) and it should not be empty.

Example

```
Begin
  if not(Growth.Recorder1.Start) then ShowMessage('Error starting recorder');
End;
```

See also: Recorder functions

Recorder1.Stop

Stops the recorder

Format

Procedure **Recorder1.stop**;

Description

This procedure stops the current record.

Example

```
Begin
  if Growth.Recorder1.Status = 1
    then Growth.Recorder1.Stop;
    else ShowMessage('Recorder was not running');
End;
```

See also: Recorder functions

Sound

Methods
- MessageBeep
- PlayAudioFile

PlayAudioFile

Plays a sound file

Format

Procedure **PlayAudioFile**(wavFileName:string ; StopCurrent:Boolean);

Parameters

- wavFileName: file name of the wav file. If no directory is specified, the application will search the file into the sound subdirectory of the program directory.
- StopCurrent: when true, if a sound is currently playing then it will be stopped and the one specified will be played. When false, if a sound is currently playing then the current sound will not be interrupted.

Description

The playaudiofile plays a single waveform sound.

See also: Sound functions

MessageBeep

Plays the default windows sound

Format

Procedure MessageBeep;

Description

The MessageBeep procedure creates a sound using the default sound defined in Windows.

See also: Sound functions

String functions

Methods
- StrToReal, RealToStr, StrToInt, IntToStr, FormatFloat, IntToHex
- Int64ToStr, Int64ToHex
- HexToFloat, FloatToHex
- ASCItoStr, StrToASCI, StrToHex
- LowerCase, UpperCase, Trim
- Pos, Copy, Length, StrReplace, Delete,
- CompareText, CompareStr, SameText
- Ord, Char

ASCItoStr

Converts readable ASCII string to a standard String

Format

Function **ASCIToStr**(Str:String) : String;

Parameters

- Str: String to convert.

Returned value

A non readable string on which special characters delimited by “<” and “>” are replaced by their ASCII code.

Description

All sub string delimited by “<” and “>” are converted by its ASCII code:

<SOH> is converted to #$1
<STX> is converted to #$2
<ETX> is converted to #$3
<EOT> is converted to #$4
<ENQ> is converted to #$5
<ACK> is converted to #$6
<DC1> is converted to #$11
<DC2> is converted to #$12
<NAK> is converted to #$15
<SYN> is converted to #$16
<DC3> is converted to #$13
<DC4> is converted to #$14
<FS> is converted to #$1C
<GS> is converted to #$1D
<RS> is converted to #$1E
<US> is converted to #$1F
<CR> is converted to #$0D
<LF> is converted to #$0A
<NULL> is converted to #$0

Example

```
begin
  ShowMessage(AsciToStr('First line <CR>Second line'));
end;
```

See also: String functions

Char

Converts a byte into a character (char type)

Format

Function **Char**(value:byte) : char;

Parameters

- ***Value***: Code ASCII of the character to convert.

Returned value

- The character of its ASCII code.

Description

The Char function converts a Value into a Char using its ASCII code. The resulting char can be used in a string of characters.

Example

```
Var
  Tab : char;
  crlf:string;
begin
  tab := Char(9);
  crlf:= Char(13)+Char(10); // or crlf:= #13+#10;
  ShowMessage('Hello'+tab+'World');  // Hello    World
  ShowMessage('Hello'+crlf+'World'); // On two lines: “Hello” and “World”
end;
```

See also: String functions

Copy

Creates a copy of part of a string

Format

Function **Copy**(Source:string; StartChar, count: byte) : string;

Parameters

- ***Source***: the source string to copy.
- ***Startchar***: the first character position beginning at 1.
- ***Count***: the number of characters to copy.

Returned value

- Part of the source string

Description

The **copy** function creates a new string from part of an existing string.
The first character of a string has index = 1.
Up to **Count** characters are copied from the **StartChar** of the **Source** string to the **returned string**.
Less than **Count** characters if the end of the **Source** string is encountered before **Count** characters have been copied.

Example

```
var
  Source, Target : string;
begin
  Source := '12345678';
  Target := Copy(Source, 3, 4);
  ShowMessage('Target : '+Target); // Target : 3456
end;
```

See also: String functions

CompareStr

Compares two strings, case-sensitive

Format

Function **CompareStr**(S1, S2 : string):integer;

Parameters

- **S1** and **S2** are two strings to compare.

Returned value

- Result of the comparison of two strings: see description.

Description

CompareText compares **S1** and **S2** and returns 0 if they are equal. If **S1 is greater than S2**, CompareText returns an integer greater than 0. If **S1 is less than S2**, CompareText returns an integer less than 0. CompareStr is case sensitive. To compare two strings regardless of case, use CompareText.

Example

```
Var
  S:string;
begin
  S := QueryStr('Enter the admin password');
  if CompareStr(S,'admin')=0 then
  Begin
```

```
    ShowMessage('Good, the password is exactly the same');
  end else ShowMessage('Failed');
end;
```

See also: String functions

CompareText

Compares two strings for equality, ignoring case

Format

Function **CompareText**(S1, S2 : string):integer;

Parameters

- **S1** and **S2** are two strings to compare.

Returned value

- Result of the comparison of two strings: see description.

Description

CompareText compares **S1** and **S2** and returns 0 if they are equal. If **S1 is greater than S2**, CompareText returns an integer greater than 0. If **S1 is less than S2**, CompareText returns an integer less than 0. CompareText is not case sensitive and is not affected by the current locale.

Example

```
Var
  S:string;
begin
  S := QueryStr('Enter the admin password');
  if CompareText(S,'admin')=0 then
  Begin
    ShowMessage('Good');
  end else ShowMessage('Failed');
end;
```

See also: String functions

Delete

Deletes a section of characters from a string

Format

Function **Delete**(source : string; StartChar, Count:byte): string;

Parameters

- Source: the source string.
- StartChar: the number of the first character to delete
- Count: the number of characters to delete.

Returned value

The new string without the deleted part.

Description

The Delete procedure deletes up to **Count** characters from the passed parameter **Source** string starting from position **StartChar**.
No error is produced if **Count** exceeds the remaining character count of **Source**.
The first character of a string is at position 1. If the **StartChar** is equal to zero, or after the last character of Source, then no characters are deleted.

Example

```
Var
  Source : string;
begin
  Source := '12345678';
  Source := Delete(Source, 3, 4);    // Delete the 3rd, 4th, 5th and 6th characters
  ShowMessage('Source now : '+Source); // Source now : 1278
end;
```

See also: String functions

FloatToHex

Converts a float into a string coded hexadecimal

Format

Function **FloatToHex**(Value:Real,Size,Format:byte) : String

Parameters

- Value: float value to convert.
- Size: number of bits of the float (32 or 64)
- Format: bitmap with:
  b0= 0 if each byte is coded into one character or b0=1 if each byte is coded into two characters
  b1= 0 if the byte order is LSB ==> MSB or b1=1 if the byte order is MSB ==> LSB

Returned value

- returned value converted into a string

Description

The FloatToHex function converts a float value into a string in which each character or group of two characters represent a byte (depending of the format).
See example bellow:

Example

```
Var
  S : string;
  R : real;
Begin

  S := FloatToHex(1.1234,32,0); // S = <#$3F><#$8F><#$CB><#$92>
  R := HexToFloat(S,32,0);
  ShowValue('R=',R); // R=1,12339997291565

  S := FloatToHex(1.1234,32,1); // S = '3F8FCB92'
  R := HexToFloat(S,32,1);
  ShowValue('R=',R); // R=1,12339997291565


  S := FloatToHex(1.1234,32,2); // S =<#$92><#$CB><#$8F><#$3F>
  R := HexToFloat(S,32,2);
  ShowValue('R=',R); // R=1,12339997291565

  S := FloatToHex(1.1234,32,3); // S = '92CB8F3F'
  R := HexToFloat(S,32,3);
  ShowValue('R=',R); // R=1,12339997291565

End;
```

See also: String functions

FormatFloat

Rich formatting of a floating-point number into a string

Format

Function **FormatFloat**(Format:String;R:real):string

Parameters

- Format: conversion format.
- R: Value to convert.

Returned value

The function returns a string which represents the real value given in parameter according to the given format.

Description

The FormatFloat function provides rich Formatting of a floating-point number Value into a string.
The Formatting string may contain a mix of freeform text and control characters:

0 : Forces digit display or 0
: Optional digit display
, : Forces display of thousands
. : Forces display of decimals
E+ : Forces signed exponent display
E- : Optional sign exponent display
; : Separator of +ve and -ve and zero values
These are best understood by looking at the sample code.

Example

```
var
  float : real;

begin
  // Set up our floating point number
  float := 1234.567;

  // Display a sample value using all of the format options

  // Round out the decimal value
  ShowMessage('##### : '+FormatFloat('#####', float));
  ShowMessage('00000 : '+FormatFloat('00000', float));
  ShowMessage('0     : '+FormatFloat('0'    , float));
  ShowMessage('#,##0 : '+FormatFloat('#,##0', float));
  ShowMessage(',0    : '+FormatFloat(',0'   , float));
  ShowMessage('');

  // Include the decimal value
  ShowMessage('0.#### : '+FormatFloat('0.####', float));
  ShowMessage('0.0000 : '+FormatFloat('0.0000', float));
  ShowMessage('');

  // Scientific format
  ShowMessage('0.0000000E+00 : '+FormatFloat('0.0000000E+00', float));
  ShowMessage('0.0000000E-00 : '+FormatFloat('0.0000000E-00', float));
  ShowMessage('#.#######E-## : '+FormatFloat('#.#######E-##', float));
  ShowMessage('');

  // Include freeform text
  ShowMessage('"Value = "0.0 : '+FormatFloat('"Value = "0.0', float));
  ShowMessage('');

  // Different formatting for negative numbers
  ShowMessage('0.0 : '+FormatFloat('0.0'              , -1234.567));
  ShowMessage('0.0 "CR";0.0 "DB" : '+
              FormatFloat('0.0 "CR";0.0 "DB"', -1234.567));
  ShowMessage('0.0 "CR";0.0 "DB" : '+
              FormatFloat('0.0 "CR";0.0 "DB"',  1234.567));
  ShowMessage('');

  // Different format for zero value
  ShowMessage('0.0 : '+FormatFloat('0.0' , 0.0));
  ShowMessage('0.0;-0.0;"Nothing" : '+
              FormatFloat('0.0;-0.0;"Nothing"', 0.0));
end;
```

Result

```
##### : 1235
   00000 : 01235
```

```
0       : 1235
#,##0 : 1,235
,0      : 1,235

0.#### : 1234.567
0.0000 : 1234.5670

0.0000000E+00 : 1.2345670E+03
0.0000000E-00 : 1.2345670E03
#.#######E-## : 1.234567E3

"Value = " : Value = 1234.6

0.0 : -1234.6
0.0 "CR";0.0 "DB" : 1234.6 DB
0.0 "CR";0.0 "DB" : 1234.6 CR

0.0 : 0.0
0.0;-0.0;"Nothing" : Nothing
```

See also: String functions

HexToFloat

Converts a String coded hexadecimal into a float

Format

Function **HexToFloat**(Value:String;Size,Format:byte) : Real
Parameters

- Value: string value to convert.
- Size: number of bits of the float represented in the string (32 or 64)
- Format: bitmap with:
  b0= 0 if each byte is coded into one character or b0=1 if each byte is coded into two characters
  b1= 0 if the byte order is LSB ==> MSB or b1=1 if the byte order is MSB ==> LSB

Returned value

- float value

Description

The HexToFloat function converts a string in which each character or group of two characters represent a byte (depending of the format) into a float.
See example bellow:

Example

```
Var
  S : string;
  R : real;
Begin

  S := FloatToHex(1.1234,32,0); // S = <#$3F><#$8F><#$CB><#$92>
  R := HexToFloat(S,32,0);
  ShowValue('R=',R);

  S := FloatToHex(1.1234,32,1); // S = '3F8FCB92'
  R := HexToFloat(S,32,1);
  ShowValue('R=',R);
```

```
  S := FloatToHex(1.1234,32,2); // S =<#$92><#$CB><#$8F><#$3F>
  R := HexToFloat(S,32,2);
  ShowValue('R=',R);

  S := FloatToHex(1.1234,32,3); // S = '92CB8F3F'
  R := HexToFloat(S,32,3);
  ShowValue('R=',R);
End;
```

See also: String functions

IntToHex

Converts an Integer into a hexadecimal string

Format

Function **IntToHex**(DecimalValue:dword; MinimumWidth:byte) : String;

Parameters

- DecimalValue: 32 bits value to convert.
- MinimumWidth: Number of digit

Returned value

- Hexadecimal string

Description

The IntToHex function converts *DecimalValue* into an hexadecimal format sting of at least *MinimumWidth* characters wide.
The resulting string has no prefix character.

Example

```
Begin
 ShowMessage('1234 decimal = '+IntToHex(1234, 1)); // 1234 decimal = 4D2
 ShowMessage('1234 decimal = '+IntToHex(1234, 8)); // 1234 decimal = 000004D2
End;
```

See also: String functions

Int64ToHex

Converts a 64 bits Integer into a hexadecimal string

Format

Function **Int64ToHex**(DecimalValue:int64; MinimumWidth:byte) : String;

Parameters

- DecimalValue: 64 bits value to convert.
- MinimumWidth: Number of digit

Returned value

- Hexadecimal string

Description

The Int64ToHex function converts *DecimalValue* number into a hexadecimal format sting of at least *MinimumWidth* characters wide.
The resulting string has no prefix character.

Example

```
Begin
 ShowMessage('1234 decimal = '+Int64ToHex(1234, 1)); // 4D2
```

```
End;
```

See also: String functions

Int64ToStr

Converts a 64 bits integer value into a string

Format

Function **Int64ToStr**(L : int64) : String

Parameters

- L: 64 bits Integer value to be converted.

Returned value

- Decimal string

Description

The IntToStr function converts an Integer Number into a string.
It is normally used for display purposes.

Example

```
Var
  BigInteger    : Int64;
begin
  BigInteger := $1000F000F000F000; // 64 bits number
  ShowMessage('BigInteger : '+Int64ToStr(BigInteger));
end;
```

See also: String functions

IntToStr

Converts an integer value into a string

Format

Function **IntToStr**(L: int32) : String

Parameters

- L: Integer value to be converted.

Returned value

- Decimal string

Description

The IntToStr function converts an Integer Number into a string.
It is normally used for display purposes.

Example

```
Var
  NormalInteger : Integer;
  BigInteger    : Int64;
begin
  NormalInteger := 2147483647;  // Largest possible Integer value
  ShowMessage('NormalInteger : '+IntToStr(NormalInteger));
  ShowMessage('Calculated number : '+IntToStr(27 * 4));
end;
```

See also: String functions

Length

Returns the number of characters in an array or string

Format

Function **Length**(Str:String):byte;

Parameters

- ***Str***: the source string.

Description

The length function returns the length of the string.
An empty string will return zero.
The maximum length of a string is 252 characters.

Example

```
Var
  S:string;
begin
  S := 'World';
  ShowValue('Length=',Length(S)); // Length=5
end;
```

See also: String functions

LowerCase

Returns a string with all letters converted to lowercase

Format

Function **LowerCase**(S:string) : String;

Parameters

- S: string to convert.

Returned value

String converted to lowercase.

Description

LowerCase returns a string with the same text as the string passed in S, but with all letters converted to lowercase.

Example

```
Const
  MyString = 'This is a TEST message';
Begin
  ShowMessage(LowerCase(MyString)); // Will display: "this is a test message"
End;
```

See also: String functions

Ord

Converts the character into its ASCII code

Format

Function **Ord**(c:char): byte;

Parameters

- ***C***: characters to convert

Returned value

- The ASCII code of c

Description

It is principally used to convert characters into their numeric equivalents.

Example

```
Begin
  ShowValue('ASCII code of A=',ord('A')); // ASCII code of A=65
end;
```

See also: String functions

Pos

Finds the position of one string in another

Format

Function **Pos**(Needle, HayStack:string): byte;

Parameters

- Needle is the searched string.
- HayStack the string in which to search.

Returned value

The position of the string or zero if not found.

Description

The Pos function finds the position of one string **Needle** within another **HayStack**.
If the string is not found, 0 is returned.
The search is case sensitive.

Example

```
begin
  ShowValue('position=',pos('best','the best software')); // Position=5
end;
```

See also: String functions

RealToStr

Converts a floating point value to a string

Format

Function **RealToStr**(Value:real;NbDec:integer) : String;

Parameters

- Value: float value to convert.
- Nbdec: number of decimal number to return in the string.

Returned value

The floating-point value converted to a string with the specified number of decimal.

Description

The RealToStr function converts a floating-point number Value into a displayable string.
The value is displayed to 15 digits of precision.
The value type may be a real type but also integer or Boolean types.
Values before the decimal point (the mantissa) exceeding the display capacity (12) result in a display using the exponent value, such as 1.2E12.

Example

```
Var
  S : string;
Begin
  S := RealToStr(1234567890123,5);
  ShowMessage(s);
End;
```

See also: String functions

SameText

Compares two strings by ordinal value without case sensitivity.

Format

Function **SameText**(S1, S2 : string):boolean;

Parameters

- **S1** and **S2** are two strings to compare.

Returned value

- True if S1 and S2 are equal.

Description

SameText compares **S1** and **S2** and returns true if they are equal.
SameText is not case sensitive.
To compare two strings in a case-sensitive way, use CompareStr.

Example

```
Var
  S:string;
begin
  S := QueryStr('Enter the admin password');
  if SameText(S,'admin')=0 then
  Begin
    ShowMessage('Good, the password is the same (without case sensitivity)');
  end else ShowMessage('Failed');
end;
```

See also: String functions

StrReplace

Replaces a part of string by another

Format

Function **StrReplace**(Source : string; position:byte; newString:string):String;

Parameters

- Source : original string
- Position: position of the first character to replace
- NewString: The string to replace into Source

Returned value

String character with the NewString

Description

The StrReplace function replaces a part of string in **Source** beginning at **Position** for the length of NewString by **NewString**.

Example

```
begin
  ShowMessage(StrReplace('Cell1.Temperature',5,'4')); // Cell4.Temperature
end;
```

See also: String functions

StrToASCI

Converts a non readable string to a readable ASCII string

Format

Function **StrToASCI**(Str:String) : String;

Parameters

- Str: String to convert.

Returned value

A readable string on which special characters are delimited by “<” and “>”

Description

All ASCII codes lower than space or greater than “~” are translated into the hexadecimal conversion of the ASCII code between “<” and “>” except for the followings ASCII code:

#$01 is converted to <SOH>
#$02 is converted to <STX>
#$03 is converted to <ETX>
#$04 is converted to <EOT>
#$05 is converted to <ENQ>
#$06 is converted to <ACK>
#$0C is converted to <CR>
#$0D is converted to <LF>
#$11 is converted to <DC1>
#$12 is converted to <DC2>
#$13 is converted to <DC3>
#$14 is converted to <DC4>
#$15 is converted to <NAK>
#$16 is converted to <SYN>
#$1C is converted to <FS>
#$1D is converted to <GS>
#$1E is converted to <RS>
#$1F is converted to <US>

Typical application: display or log a communication frame.

Example

```
begin
  ShowMessage(StrToASCI(#2+'ABC'+#3+#13+#10));//will display: <STX>ABC<ETX><$0D><$0A>
end;
```

See also: String functions

StrToHex

Converts each character in a string to a sequence of hexadecimal numbers

Format

Function **StrToHex**(Str, Separator:String) : String;

Parameters

- Str: String to convert.
- Separator: Characters inserted between two hexadecimal numbers.

Returned value

Returns a string made up of a series of bytes displayed in ASCII coded hexadecimal and separated by the separator parameter.

Description

The string 'ABC' will be converted to '41, 42,43 'if the separator is a comma.
The string '0123' will be converted to '30; 31; 32; 33 'if the separator is'; '

Example

```
begin
  ShowMessage(StrToHex(#2+'ABC'+#3+#13+#10,',','));//will display: 02,41,42,43,03,13,10
end;
```

See also: String functions

StrToInt

Converts an integer string into an Integer value

Format

Function **StrToInt**(IntegerString: String): integer;

Parameters

- IntegerString: String value to be converted.

Returned value

- Integer value

Description

The StrToInt function converts an Integer string, IntegerString such as '123' into an Integer.
It supports hexadecimal numbers, as prefixed by $.
In case of error, the function returns -999999

Example

```
var
  A, B, C, D, E : Integer;
begin
  A := 32;
  B := StrToInt('100');    // '100' string converted to 100 integer
  C := StrToInt('  -12');  // Leading blanks are ignored
  D := StrToInt('$1E');    // Hexadecimal values start with a '$'
  E := A + B + C + D ;     // Lets add up all these integers

  ShowMessage('A : '+IntToStr(A)); // A : 32
  ShowMessage('B : '+IntToStr(B)); // B : 100
  ShowMessage('C : '+IntToStr(C)); // C : -12
  ShowMessage('D : '+IntToStr(D)); // D : 30
  ShowMessage('E : '+IntToStr(E)); // E : 150
end;
```

See also: String functions

StrToReal

Converts a number string into a floating point value

Format

Function **StrToReal**(FloatString:string; Trunc:boolean) : real;

Parameters

- FloatString: number string to convert.
- Trunc: true to return only the integer value.

Returned value

The floating point value.

Description

The StrToReal function converts a number string, *FloatString* such as '123.456' into a floating point number. It supports integer, floating point, and scientific (exponent) formats.
If the function failed, the return value will be: -999999

Example

```
Var
  R : real;
Begin
  R := StrToReal('12.34',false);
  ShowValue('R=',R);
End;
```

See also: String functions

Trim

Trims leading and trailing spaces and control characters from a string

Format

Function **Trim**(S:string) : String;

Parameters

- S: source string.

Returned value

String without blank and control characters (such as line feed) from the start and end of a source string.

Description

Trim removes leading and trailing spaces and control characters from the given string S.
In the 7-bit ASCII character set defined in ANSI X3.4-1977 (C0 and G0), "control codes" are defined as all characters whose code is between 0 and 31.

Example

```
Const
  MyString = '   This is a TEST message '+#13+#10;
Begin
  ShowMessage(Trim(MyString)); // Will display: "This is a TEST message"
End;
```

See also: String functions

UpperCase

Returns a string with all letters converted to uppercase

Format

Function **UpperCase**(S:string) : String;

Parameters

- S: string to convert.

Returned value

String converted to Uppercase.

Description

UpperCase returns a string with the same text as the string passed in S, but with all letters converted to uppercase.

Example

```
Const
  MyString = 'This is a TEST message';
Begin
  ShowMessage(UpperCase(MyString)); // Will display: "THIS IS A TEST MESSAGE"
End;
```

See also: String functions

Sub Equipment

See also

- Equipment functions

GetTagColumn

Popup the list view

Format

Function **GetTagColumn**(TagName,ColumnName:string):string;

Parameters

- TagName: Name of the tag
- ColumnName: Name of the column (see description)

Returned value

- The content of the cell identified by TagName and ColumnName

Description

This function is used to read the table of configuration of a sub-equipment.
The parameter ColumnName can be:
Type
Regulator tag
User name
Min
Max
Comment
Opt
Recipe
Default
Chart
OnWrite

Example

```
Begin
  ShowMessage(Growth.Cell1.Temperature.GetTagColumn('MV','Max'));
End;
```

See also: Equipment methods

SMS

Method

- SendSMS

SendSMS

Sends a short text message

Format

Function **SendSMS** (Message:String):boolean; // return true if the SMS is enabled

Parameters

- Message: Text message to send

Returned value

Returns true if successful

Description

The function SendSMS allows sending SMS to recipients defined in the options form.
The SMS option must be enabled in the options form otherwise the function won't have any effect and will return false.
Depending of the configuration, the sending of the SMS may be delayed. In the options check the parameters "Maximum delay before sending a SMS" and "Number of messages before sending a SMS".

Example

```
Var
  Msg : String;
Begin
  Msg := 'IMPORTANT'+#13;
  Msg := Msg+'Al cell temperature is ='+RealToStr(Growth.Al_ANB6_8_P6.temperature.MV,1);
  SendSMS(Msg);
End;
```

System functions

Methods

- ClearConsole
- Exit
- LogEvent
- RunEvent
- ShellExecute
- Sleep
- StopOnError
- WinExec
- WriteConsole

ClearConsole

Clears the content of the console

Format

Procedure **ClearConsole**;

Description

Clears the content of the console.

Example

```
Begin
  ClearConsole;
End;
```

See also: System functions

Exit

Stops the execution of the script without any message

Format

Procedure **Exit**;

Example

```
begin
  if QueryYesNo('Do you want to stop now ?') then Exit;
  ShowMessage('The script continue');
end;
```

See also: System functions

LogEvent

Adds an event into the log list and log file

Format

Procedure **LogEvent**(ChNum, LogIdent, LogType, ErrLevel : integer ; Message : String);

Parameters

- ***ChNum*** is the chamber number or zero
- ***LogIdent*** is an identifier of the log message (790 for user script)
- ***LogType***: 0 = LOG_GENERAL, 1 = LOG_COM, 2 = LOG_ALARM, 3 = LOG_USER
- ***ErrLevel***: 0 = EL_NONE, 1 = EL_GOOD, 2 = EL_WARNING, 3 = EL_CRITICAL, 4 = EL_RESTORED
- ***Message***: Message of the event

Description

This procedure can be used to add an event in the log list.

Example

```
Begin
  LogEvent(0, 790, 2, 3,'No water flow, the process cannot continue');
End;
```

See also: System functions

RunEvent

Executes an event

Format

Procedure **RunEvent**(Component,Event : string);

Parameters

- ***Component***: Name of a component
- ***Event***: name of the Event

Description

This procedure allows running an event of a component.
By example, you can simulate a click of mouse on a button by calling its events OnClick.

Example

```
Begin
  RunEvent('BitBtn1','OnClick');
End;
```

See also: System functions

ShellExecute

Executes the specified application

Format

Function **ShellExecute**(Operation,FileName,Parameters,Directory:String;CmdShow:word):word

Parameters

- **Operation**
  'edit' = Launches an editor and opens the document for editing
  'explore' = Explores a folder
  'find' = Initiates a search beginning in the directory
  'open' = Opens an item, file, folder, link
  'print' = Prints the file
- **Filename**
  FileName, link URL to open and modify
- **Parameters**
  The parameters to be passed to the application delimited by space. Ex - '-c -i -v'
- **Directory**
  The default (working) directory for the action.
- **CmdShow**
  Specifies how the window is to be shown. This parameter can be one of the following values.

  SW_SHOWNORMAL (1): Activates and displays a window.

  SW_FORCEMINIMIZE (11): Windows 2000/XP: Minimizes a window, even if the thread that owns the window is not responding.
  SW_HIDE (0): Hides the window and activates another window.
  SW_MAXIMIZE (3): Maximizes the specified window.
  SW_MINIMIZE (6): Minimizes the specified window and activates the next top-level window in the Z order.
  SW_RESTORE (9): Activates and displays the window. If the window is minimized or maximized, the system restores it to its original size and position.
  SW_SHOW(5): Activates the window and displays it in its current size and position.
  SW_SHOWMINIMIZED(2): Activates the window and displays it as a minimized window.
  SW_SHOWMINNOACTIVE(7): Displays the window as a minimized window. This value is similar to SW_SHOWMINIMIZED, except the window is not activated.
  SW_SHOWNA (8): Displays the window in its current size and position. This value is similar to SW_SHOW, except the window is not activated.
  SW_SHOWNOACTIVATE (4): Displays a window in its most recent size and position. This value is similar to SW_SHOWNORMAL, except the window is not actived.

Returned value

If the return value of ShellExecute is greater than 32, the application was executed successfully.
If its less or equal than 32 then the function failed.
Here is a complete list of the possible return values of ShellExecute:

- 0 = The operating system is out of memory or resources.
- 2 = The specified file was not found
- 3 = The specified path was not found.
- 5 = Windows 95 only: The operating system denied access to the specified file
- 8 = Windows 95 only: There was not enough memory to complete the operation.

- 10 = Wrong Windows version
- 11 = The .EXE file is invalid (non-Win32 .EXE or error in .EXE image)
- 12 = Application was designed for a different operating system
- 13 = Application was designed for MS-DOS 4.0
- 15 = Attempt to load a real-mode program
- 16 = Attempt to load a second instance of an application with non-readonly data segments.
- 19 = Attempt to load a compressed application file.
- 20 = Dynamic-link library (DLL) file failure.
- 26 = A sharing violation occurred.
- 27 = The filename association is incomplete or invalid.
- 28 = The DDE transaction could not be completed because the request timed out.
- 29 = The DDE transaction failed.
- 30 = The DDE transaction could not be completed because other DDE transactions were being processed.
- 31 = There is no application associated with the given filename extension.
- 32 = Windows 95 only: The specified dynamic-link library was not found.

Description

ShellExecute is Windows API function that is mostly used for launch external applications. The function returns an integer that corresponds to an error code which is very useful when we need to show some status if the function worked or not.

Example

```
Begin
  FName := App.ProjectDirectory+'Template\MyWordTemplate.docm';
  // Open the word document
  res :=ShellExecute('open',FName,'','',1);
  if res>32 then ShowMessage('ShellExecute executed with success');
End;
```

See also: System functions

Sleep

Pauses script execution for a specified number of milliseconds.

Format

Procedure **Sleep**(delay:real);

Parameters

- ***Delay***: time in milliseconds.

Description

This function pauses the execution of the script for the specified **delay**.

Example

```
Begin
  ShowMessage('Script is running');
  sleep(3000);
  ShowMessage('end of script');
End;
```

See also: System functions

StopOnError

Stops the script with an error code

Format

Procedure **StopOnError**(ErrCode: int32;ErrMsg:String;ShowError :boolean)

Parameters

- ***ErrorCode***: the error number to display and log.
- ***ErrMsg***: The error message to display and to log.
- ***ShowError***: = True to display the error message and to log
  = False to stop without any information (same as EXIT)

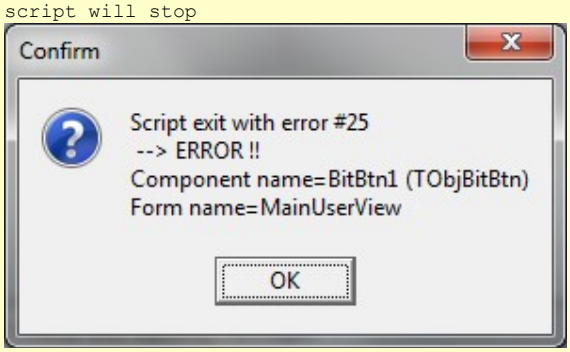
Description

This procedure displays an error message and stop the execution of the script.

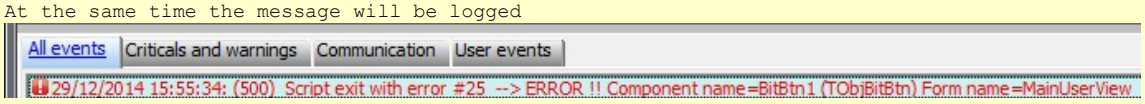
Example

```
begin
  if QueryYesNo('Do you want to stop now ?') then StopOnError(25,'ERROR !!',false);
  ShowMessage('The script continue');
end;
```

In this example, if the user push on YES then the following message will be displayed and the script will stop



At the same time the message will be logged



See also: System functions

WinExec

Runs the specified application

Format

Function **WinExec**(CmdLine : string ; CmdShow : word): word

Parameters

- **CmdLine:**
  The command line (file name plus optional parameters) for the application to be executed. If the name of the executable file does not contain a directory path, the system searches for the executable file in this sequence:
  - The directory from which the application loaded.
  - The current directory.
  - The Windows system directory.
  - The Windows directory.
  - The directories listed in the PATH environment variable.

- **CmdShow:**

Specifies how the window is to be shown. This parameter can be one of the following values.

SW_SHOWNORMAL (1): Activates and displays a window.

SW_FORCEMINIMIZE (11): Windows 2000/XP: Minimizes a window, even if the thread that owns the window is not responding.
SW_HIDE (0): Hides the window and activates another window.
SW_MAXIMIZE (3): Maximizes the specified window.
SW_MINIMIZE (6): Minimizes the specified window and activates the next top-level window in the Z order.
SW_RESTORE (9): Activates and displays the window. If the window is minimized or maximized, the system restores it to its original size and position.
SW_SHOW(5): Activates the window and displays it in its current size and position.
SW_SHOWMINIMIZED(2): Activates the window and displays it as a minimized window.
SW_SHOWMINNOACTIVE(7): Displays the window as a minimized window. This value is similar to SW_SHOWMINIMIZED, except the window is not activated.
SW_SHOWNA (8): Displays the window in its current size and position. This value is similar to SW_SHOW, except the window is not activated.
SW_SHOWNOACTIVATE (4): Displays a window in its most recent size and position. This value is similar to SW_SHOWNORMAL, except the window is not actived.

Returned value

If the function succeeds, the return value is greater than 31.
If the function fails, the return value is one of the following error values:

- 0 : The system is out of memory or resources
- ERROR_BAD_FORMAT (11): The .exe file is invalid.
- ERROR_FILE_NOT_FOUND (2): The specified file was not found.
- ERROR_PATH_NOT_FOUND (3): The specified file was not found.

Description

Runs the specified application.
This function executes the windows API WinExec function.

Example

```
Begin
  WinExec('Notepad.exe',1);
End;
```

See also: System functions

WriteConsole

Outputs a message to the console

Format

Procedure **WriteConsole**(Text:string)

Parameters

- ***Text***: text to print to the console.

Description

This function outputs a text to the console. The text will be preceded by the date and the time.
If the console is closed then it will be opened automatically.
This procedure can be used to debug a script but it is not recommended to use it in a released version.

Example

```
Begin
  Writeconsole('Hello world');
End;
```

See also: System functions

Timer

Methods

- GetTimer
- ResetTimer
- Pausetimer
- ResumeTimer

GetTimer

Returns the time elapsed of a timer

Format

Function **GetTimer**(TimerId:byte) : real;

Parameters

- TimerId: Number of the timer from 1 to 10

Returned value

- Time in second

Description

The GetTimer function returns the elapsed time since the function ResetTimer has been called.
Up to 10 timers can be used in a script.
All timers are local to the script and then are not accessible from another script.

Example

```
Begin
  ResetTimer(1);
  ShowMessage('Press Ok to continue');
  ShowValue('Time elapsed',GetTimer(1));
end;
```

See also: Timer functions

ResetTimer

Resets a timer

Format

Function **ResetTimer**(TimerId:byte): real;

Parameters

- TimerId: Number of the timer from 1 to 10

Returned value

- Time in second

Description

The ResetTimer function Restart the timer identified by TimerId.
Up to 10 timers can be used in a script.
All timers are local to the script and then are not accessible from another script.

Example

```
Begin
  ResetTimer(1);
  ShowMessage('Press Ok to continue');
  ShowValue('Time elapsed',GetTimer(1));
end;
```

See also: Timer functions

PauseTimer

Suspends a timer

Format

Procedure **PauseTimer**(TimerId:byte);

Parameters

- TimerId: Number of the timer from 1 to 10

Description

The PauseTimer function suspends the timer identified by TimerId.
To resume, call the ResumeTimer function with the same id.
Up to 10 timers can be used in a script.
All timers are local to the script and then are not accessible from another script.

Example

```
Begin
  ResetTimer(1);
  ShowMessage('Press Ok to continue');
  Pausetimer(1);
  ShowValue(' Now the timer is paused - Time elapsed',GetTimer(1));
  ResumeTimer(1);
  ShowValue('The timer is resumed, Time elapsed',GetTimer(1));
end;
```

See also: Timer functions

ResumeTimer

Resumes a timer

Format

Procedure **ResumeTimer**(TimerId:byte);

Parameters

- TimerId: Number of the timer from 1 to 10

Description

The ResumeTimer function resumes the timer identified by TimerId.
This procedure must be call after the call of PauseTimer procedure.
Up to 10 timers can be used in a script.
All timers are local to the script and then are not accessible from another script.

Example

```
Begin
  ResetTimer(1);
  ShowMessage('Press Ok to continue');
  Pausetimer(1);
  ShowValue(' Now the timer is paused - Time elapsed',GetTimer(1));
  ResumeTimer(1);
  ShowValue('The timer is resumed, Time elapsed',GetTimer(1));
end;
```

See also: Timer functions

User functions

User methods must be preceded by “app.user.”

Methods

- App.User.Authorized_Chamber
- App.User.Authorized_Right

Authorized_Chamber

(**App.User.Authorized_Chamber** method)

Returns true if the current user is allowed for this chamber

Format

Function Autorized_Chamber(chName:string):boolean;

Parameters

- ChName: Name of the chamber

Description

This function is used to know if the current user is allowed for a specific chamber.
The parameter *chName* indicates which chamber to check.

Example

```
uses StdConst;
Begin
  if not(App.User.Autorized_Chamber('Growth1')) then EXIT;
  ShowMessage('This message is displayed because you are allowed to continue');
End;
```

See also: User functions

Authorized_Right

(**App.User.Authorized_Right** method)

Returns true if the current user is allowed for this right and chamber

Format

Function Autorized_Right(Right:TRights;chName:string;ShowMsg:boolean):boolean;

Parameters

- Right : a constant defined in StdConst (see description for more detail)
- ChName: Name of the chamber or an empty string for any chamber
- ShowMsg: true to display a message if the user is not allowed.

Description

This function is used to know if the current user is allowed for a specific operation.
The parameter *Right* indicates which right to check.
If the parameter *ChName* is empty it will be ignored.
Add the keyword Uses StdConst at the beginning of the script to use a specific constants.
Here is the list of constact that can be used:

```
usr_rt_HW              // 'Hardware configuration'
usr_rt_Options          // 'Options'
usr_rt_Users            // 'User rights'
usr_rt_file_new         // 'File New'
usr_rt_file_open        // 'File open'
usr_rt_Equipment        // 'Control equipment'
usr_rt_Setup            // 'Setup equipment'
usr_rt_DisableCom       // 'Disable com & sockets'
usr_rt_Sec_OnOff        // 'Securities On/Off'
usr_rt_Sec_Cfg          // 'Securities change'
usr_rt_Edit_forms       // 'Edit forms'
usr_rt_Edit_script      // 'Edit script'
usr_rt_Edit_template    // 'Edit templates'
usr_rt_recipe           // 'Recipe control'
```

```
usr_rt_batch            // 'Batch control'
usr_rt_recorder         // 'Recorder control'
usr_rt_acknowledge_alarms // 'Acknowledge alarms'
usr_rt_ExitApp          // 'Exit application'
```

Example

```
uses StdConst;
Begin
  if not(App.User.Autorized_Right(usr_rt_HW,'',true)) then EXIT;
  ShowMessage('This message is displayed because you are allowed to continue');
End;
```

See also: User functions

XML methods

Methods

- XML_Clear
- XML_Create
- XML_CreateGlobal
- XML_LoadFromFile
- XML_SaveToFile
- XML_GetChildCount
- XML_Read
- XML_Write
- XML_Answer (ASCII server only)
- XML_ReceiveBuf (ASCII server only)
- XML_Send (ASCII module/device only)

XML_Clear

Delete an XML node or attribute

Format

Function XML_Clear(IdXML:integer; Path,Item:string):Boolean;

Parameters

- IdXML: XML document identifier (see XML_Create and XML_CreateGlobal function)
- Path: String path specifying the node (separated by "/")
- Item: Attribute of the node or blank to delete the node.

Returned value

The function returns true if success.

Description

Use XML_Clear if you want to remove a node or an attribute in an XML document.
Specify the node by the path and the item parameters. If no item (attribute) parameter is entered (empty string) then the entire node will be deleted.

For global xml, the path is case sensitive (except the beginning of the path including the document name).
For local xml, the path is not case sensitive.

Example

```
Var
 Idx : integer;
Begin
  Idx := XML_Create;
  XML_Write(idx,'xml','','');
  XML_Write(idx,'xml/Alarm','Name','OverTemperature');
  XML_Write(idx,'xml/Alarm','Time',FormatDateTime('YYYY-MM-DD"T"HH:MM:SS',GetNow));
  XML_Write(idx,'xml/Alarm','Severity','critical');
  XML_Write(idx,'xml/Alarm','IsSet','true');
  XML_Write(idx,'xml/Alarm/V','name','Temperature');
  XML_Write(idx,'xml/Alarm/V','','2300');
  XML_Clear(idx,'xml/Alarm/V','');
  XML_SaveToFile(idx,'c:\temp\alarm.xml');
```

```
End;
```

See also: XML functions

XML_Create

Creates an local XML document (which is only accessible in the current script).

Format

Function XML_Create:Integer;

Returned value

The function returns an XML document identifier.

Description

Creates an XML document and returns the identifier that can be use with other XML functions.
The XML document is only reachable in the current script.
To access an global XML document, use the function XML_CreateGlobal.

Example

```
Var
 Idx : integer;
Begin
  Idx := XML_Create;
  XML_Write(idx,'xml','','');
  XML_Write(idx,'xml/Alarm','Name','OverTemperature');
  XML_Write(idx,'xml/Alarm','Time',FormatDateTime('YYYY-MM-DD"T"HH:MM:SS',GetNow));
  XML_Write(idx,'xml/Alarm','Severity','critical');
  XML_Write(idx,'xml/Alarm','IsSet','true');
  XML_Write(idx,'xml/Alarm/V','name','Temperature');
  XML_Write(idx,'xml/Alarm/V','','2300');
  XML_SaveToFile(idx,'c:\temp\alarm.xml');
End;

Resulting output file:
<xml>
  <Alarm Name="OverTemperature" Time="2018-03-29T09:03:58" Severity="critical" IsSet="true">
    <V name="Temperature">2300</V>
  </Alarm>
</xml>
```

See also: XML functions

XML_CreateGlobal

Creates or access a global xml document (which is accessible from all scripts).

Format

Function XML_CreateGlobal(DocumentName:string):integer;

Returned value

The function returns an XML document identifier which can be used by all XML functions.

Description

Creates an Global XML document and returns the identifier that can be use with other XML functions.
If the document name already exists, the function returns an identifier on the existing document.

The global document is automatically saved in the data.dat file regularly and when exiting the application.
The document is automatically loaded when the application starts.
It is possible to edit the document by using the integrated xml editor from the main menu **Tools/XML Viewer.**
Global XML documents is a good solution for sharing data between multiple scripts.

Example

```
Var
 Idx : integer;
Begin
  Idx := XML_CreateGlobal('User');
  XML_Write(idx,'xml','','');
  XML_Write(idx,'xml/Alarm','Name','OverTemperature');
  XML_Write(idx,'xml/Alarm','Time',FormatDateTime('YYYY-MM-DD"T"HH:MM:SS',GetNow));
  XML_Write(idx,'xml/Alarm','Severity','critical');
  XML_Write(idx,'xml/Alarm','IsSet','true');
  XML_Write(idx,'xml/Alarm/V','name','Temperature');
  XML_Write(idx,'xml/Alarm/V','','2300');
  XML_SaveToFile(idx,'c:\temp\alarm.xml');
End;

Resulting output file:
<xml>
  <Alarm Name="OverTemperature" Time="2018-03-29T09:03:58" Severity="critical" IsSet="true">
    <V name="Temperature">2300</V>
  </Alarm>
</xml>
```

See also: XML functions

XML_LoadFromFile

Loads an XML file

Format

Function XML_LoadFromFile(IdXML:integer; Filename:string):Boolean;

Parameters

-IdXML: XML document identifier (see XML_Create and XML_CreateGlobal function)
-**FileName**: Path and name of the file to load.

Returned value

The function returns true if the file as been successfully loaded.

Description

This function loads an XML document identified by IdXML and FileName and returns true if the file as been successfully loaded.

Example

```
Var
  Idx : integer;
Begin
  Idx := XML_Create;
  ClearConsole;
  if XML_LoadFromFile(idx,'c:\temp\alarm.xml') then
  Begin
    WriteConsole('Root document is :'+XML_Read(idx,'#root','','item not found'));
    WriteConsole('First node is :'+XML_Read(idx,'#0','','item not found'));
    WriteConsole('Name of alarm is: '+XML_Read(idx,'xml/Alarm/V','name','item not found'));
  end else WriteConsole('Error reading XML file');
End;

Read file:
<xml>
  <Alarm Name="OverTemperature" Time="2018-03-29T09:03:58" Severity="critical" IsSet="true">
    <V name="Temperature">2300</V>
  </Alarm>
</xml>
```

See also: XML functions

XML_SaveToFile

Saves an XML document to a file

Format

Function XML_SaveToFile(IdXML:integer; Filename:string):Boolean;

Parameters

-IdXML: XML document identifiant (see XML_Create and XML_CreateGlobal function)
-**FileName**: Path and name of the file to save.

Returned value

The function returns true if the file as been successfully saved.

Description

This function saves the XML document identified by IdXML into the file Filename and returns true if the file as been successfully saved.

Example

```
Var
  Idx : integer;
Begin
  Idx := XML_Create;
  XML_Write(idx,'xml','','');
  XML_Write(idx,'xml/Alarm','Name','OverTemperature');
  XML_Write(idx,'xml/Alarm','Time',FormatDateTime('YYYY-MM-DD"T"HH:MM:SS',GetNow));
  XML_Write(idx,'xml/Alarm','Severity','critical');
  XML_Write(idx,'xml/Alarm','IsSet','true');
  XML_Write(idx,'xml/Alarm/V','name','Temperature');
  XML_Write(idx,'xml/Alarm/V','','2300');
  XML_SaveToFile(idx,'c:\temp\alarm.xml');
End;

Resulting output file:
<xml>
  <Alarm Name="OverTemperature" Time="2018-03-29T09:03:58" Severity="critical" IsSet="true">
    <V name="Temperature">2300</V>
  </Alarm>
</xml>
```

See also: XML functions

XML_GetChildCount

Returns the number of child nodes

Format

Function XML_GetChildCount(idXML:integer; Path: String): Integer

Parameters

-IdXML: XML document identifiant (see XML_Create and XML_CreateGlobal function)
-**Path**: Path to reach the node.

Returned value

The function returns the number of child from a node identified by Path.

Description

The function returns the number of child of a node identified by Path.
The XML document is identified by idXML.

Example

```
Input XML file (alarm.xml):

<xml>
```

```
  <Alarm Name="OverTemperature" Time="2018-03-29T09:03:58" Severity="critical" IsSet="true">
    <V name="Temperature">2300</V>
    <V name="Temperature">20</V>
    <V name="Temperature">330</V>
    <V name="Temperature">150</V>
  </Alarm>
</xml>

Script:
Var
  idx,n,i : integer;
  NodeName : String;
Begin
  idx := XML_Create;
  clearConsole;
  if XML_LoadFromFile(idx,'c:\temp\alarm.xml') then
  Begin
    n := XML_GetChildCount(idx,'Alarm');
    WriteConsole('Number of nodes in Alarm:'+IntToStr(n));
    for i:=0 to n-1 do
    Begin
      NodeName := XML_Read(idx,'Alarm/#'+IntToStr(i),'name','Error');
      WriteConsole('Node #'+IntToStr(i)+': '+NodeName);
    end;
  end else WriteConsole('Error reading XML file');
End;

Console output:

Number of nodes in Alarm:4
Node #0: TempA
Node #1: TempB
Node #2: TempC
Node #3: TempD
```

See also: XML functions

XML_Read

Reads an attribute or the text of a node

Format

Function XML_Read(idXML:integer; Path,item, default: String): String;

Parameters

-IdXML: XML document identifiant (see XML_Create and XML_CreateGlobal function)
-**Path**: String path specifying the node (separated by "/")
-**Item**: Attribute of the node or blank to read the text node
-**Default**: Default string to return in case of error (path does not exist or attribute not found)

Returned value

The function returns the attribute of the text of a node.

Description

The function returns the attribute of the text of a node in the XML document idenfied by IdXML.

Example : enumerate each node of an XML file

```
XML file content (alarm.xml):

<xml>
  <Alarm Name="OverTemperature" Time="2018-03-29T09:03:58" Severity="critical" IsSet="true">
    <V name="Temperature">2300</V>
    <V name="Temperature">20</V>
    <V name="Temperature">330</V>
    <V name="Temperature">150</V>
  </Alarm>
</xml>

Script:
Var
```

```
  idx,n,i : integer;
  NodeName : String;
Begin
  idx := XML_Create;
  clearConsole;
  if XML_LoadFromFile(idx,'c:\temp\alarm.xml') then
  Begin
    n := XML_GetChildCount(idx,'Alarm');
    WriteConsole('Number of nodes in Alarm:'+IntToStr(n));
    for i:=0 to n-1 do
    Begin
      NodeName := XML_Read(idx,'Alarm/#'+IntToStr(i),'name','Error');
      WriteConsole('Node #'+IntToStr(i)+': '+NodeName);
    end;
  end else WriteConsole('Error reading XML file');
End;
```

See also: XML functions

XML_Write

Writes an attribute or the text of a node

Format

Function XML_Write(IdXML:integer; Path,item,value :string):Boolean;

Parameters

- IdXML: XML document identifiant (see XML_Create and XML_CreateGlobal function)
- **Path**: String path specifying the node (separated by "/")
- Item: Attribute of the node or blank to write the text of the node.
- **Value**: String to write

Returned value

The function returns true if the attribute or the text has been successfully written to the XML document.

Description

This function writes an attribute or the text of a node identified by the path in the XML document itself identified by IdXML.
The path must start with the first level element.
Only one top-level element is allowed in an xml document.
The path can contain key words like #n (n is an integer number which represent the number of the node).

Example 1: create a new path with an attribute and a value

```
Script

Var
  idTx : integer;
Begin
  idTx := XML_Create;
  XML_Write(idTx,'root/test1/Tag','test1','123');
  XML_SaveToFile(idTx,'c:\temp\output.xml');
End;

Resulting output file
<root>
  <test1>
    <Tag test1="123"/>
  </test1>
</root>
```

Example 2: create a new path with two attributes test1 and test2

```
Script

Var
  idTx : integer;
Begin
  idTx := XML_Create;
  XML_Write(idTx,'root/test1/Tag','test1','123');
```

```
  XML_Write(idTx,'root/test1/Tag','test2','12');
  XML_SaveToFile(idTx,'c:\temp\output.xml');
End;

Resulting output file
<root>
  <test1>
    <Tag test1="123" test2="12"/>
  </test1>
</root>
```

Example 3: create a new path with two nodes Tag

```
Script

Var
  idTx : integer;
Begin
  idTx := XML_Create;
  XML_Write(idTx,'root/test1/Tag','test1','123');
  XML_Write(idTx,'root/test1/+Tag','test2','12');
  XML_SaveToFile(idTx,'c:\temp\output.xml');
End;

Resulting output file
<root>
  <test1>
    <Tag test1="123"/>
    <Tag test2="12"/>
  </test1>
</root>
```

Example 4: read an existing XML file and change the text of the second node

```
Original XML file (alarm.xml):

<xml>
  <Alarm Name="OverTemperature" Time="2018-03-29T09:03:58" Severity="critical" IsSet="true">
    <V name="Temperature">2300</V>
    <V name="Temperature">20</V>
    <V name="Temperature">330</V>
    <V name="Temperature">150</V>
  </Alarm>
</xml>

Script:

Var
  idx: integer;
Begin
  idx := XML_Create;
  clearConsole;
  if XML_LoadFromFile(idx,'c:\temp\alarm.xml') then
  Begin
    XML_Write(idx,'xml/Alarm/#1','','111');
    XML_SaveToFile(idx,'c:\temp\alarm.xml');
  end else WriteConsole('Error reading XML file');
End;

Resulting output file after script execution:
<xml>
  <Alarm Name="OverTemperature" Time="2018-03-29T09:03:58" Severity="critical" IsSet="true">
    <V name="Temperature">2300</V>
    <V name="Temperature">111</V>
    <V name="Temperature">330</V>
    <V name="Temperature">150</V>
  </Alarm>
</xml>
```

See also: XML functions

XML_Answer

(USED BY ASCII SERVER ONLY)

Sends an XML frame to the client

Format

Function XML_Answer(IdXML:integer; Header :string):Boolean;

Parameters

- IdXML: XML document identifier (see XML_Create and XML_CreateGlobal function)
- **Header**: String added to the frame before the XML document or '@' to generate automatically the header (see description for more details)

Returned value

The function returns true if successfully completed.

Description

Use the function XML_Answer after receiving a frame from the host computer which is connected to the ASCII Server.
If the header parameter is defined, then it will be added to the frame just before the XML document.

If the header is equal to the special character **'@'** then a special header will be added.
This special header will be composed of these lines:

- Line1: <Prolog> <MessageType>
- Line2: Content-length: <len>
- Line3: Content-type: <contentType>
- Line4: Message-ID:<MessageId>
- Line5: <TransactionID>
- Line6: <Empty line>

Each line is teminated by <CR><LF>

Special key words used above are the following:
<Prolog> A copy of the prolog received in the requested frame ('XPTP/1.0')
<MessageType>:
if the received MessageType is 'DATA-REQUEST' then the MessageType sent will be 'DATA-RESPONSE'
if the received MessageType is 'PING-REQUEST' then the MessageType sent will be 'PING-RESPONSE'
<len> if the length of the XML document
<ContentType> A copy of the ContentType received in the requested frame ('text/xml; utf-8')
<MessageId> A copy of the MessageId received in the requested frame

Properties that can be used are:
- Prolog
- MessageType
- ContentType
- ContentLength
- MessageIdRx
- MessageIdTx
- TransactionId

These properties are only accessible in the script of the ASCII server.

Example of script in the ASCII server

```
Var
  idRx,idTx,NodeNum : integer;
  S : string;
//---------------------------------------------------------------------
Procedure AddOneVar(SName,SType,STranscient:string);
Var Path : String;
Begin
  XML_Write(idTx,'Variable-SD/+Variable','name',SName);
  Path := 'Variable-SD/#'+IntToStr(NodeNum);
  XML_Write(idTx,Path,'Type',SType);
  XML_Write(idTx,Path,'Transcient',STranscient);
  NodeNum := NodeNum + 1;
end;
//---------------------------------------------------------------------
Procedure SendVariablesList;
Begin
  idTx := XML_Create;
  NodeNum := 0;
  AddOneVar('Clock','TimeStamp','false');
  AddOneVar('Var2','float','false');
  AddOneVar('Var3','int','true');
  XML_Answer(idTx,'@');
```

```
End;
//----------------------------------------------------------------------
Procedure AddOneAlarm(AName,SDescription,VName:string);
Var Path : String;
Begin
  XML_Write(idTx,'Alarm-SD/+Alarm','name',AName);
  Path := 'Alarm-SD/#'+IntToStr(NodeNum);
  XML_Write(idTx,Path+'/Description','',SDescription);
  if length(VName)>0 then XML_Write(idTx,Path+'/Variable','Name',VName);
  NodeNum := NodeNum + 1;
End;
//----------------------------------------------------------------------
Procedure SendAlarmsList;
Begin
  idTx := XML_Create;
  NodeNum := 0;
  AddOneAlarm('EquipmentOnFire','Equipment environment critically overheated','Temperature');
  AddOneAlarm('Motor_3_failure','Conveyor motor 3 electrical or mechanical failure.','');
  XML_Answer(idTx,'@');
End;
//----------------------------------------------------------------------
Procedure SendRecipesList;
Begin
  idTx := XML_Create;
  XML_Write(idTx,'RecipeList-Response/+Recipe','','Ma recette 1');
  XML_Write(idTx,'RecipeList-Response/+Recipe','','Ma recette 2');
  XML_Write(idTx,'RecipeList-Response/+Recipe','','Ma recette 3');
  XML_Answer(idTx,'@');
End;
//----------------------------------------------------------------------
Procedure SendPingResponse;
Begin
  idTx := XML_Create;
  MessageType := 'PING-RESPONSE';
  XML_Answer(idTx,'@');
End;
//----------------------------------------------------------------------
Begin
  idRx := XML_Create;
  if XML_ReceiveBuf(idRx) then
  Begin
    if compareText('DATA-REQUEST',MessageType)=0 then
    Begin
      S := XML_Read(idRx, '', '','?');
      if (CompareText('Variable-SD-Query',S)=0)  then SendVariablesList;
      if (CompareText('Alarm-SD-Query',S)=0)     then SendAlarmsList;
      if (CompareText('RecipeList-Request',S)=0) then SendRecipesList;
    end;
    if compareText('PING-REQUEST',MessageType)=0 then SendPingResponse;
  End;
End;
```

See also: XML functions, ASCII server module

XML_ReceiveBuf

(USED BY ASCII SERVER ONLY)

Assign the received string into an XML document

Format

Function XML_ReceiveBuf(IdXML:integer):Boolean;

Parameters

- IdXML: XML document identifier (see XML_Create and XML_CreateGlobal function)

Returned value

The function returns true if successfully completed.

Description

Use the function XML_ReceiveBuf function after receiving a frame from the host computer which is connected to the ASCII Server.
All characters before the first character "<" are deleted.

Example of script in the ASCII server

```
Var
  idRx,idTx,NodeNum : integer;
  S : string;
//----------------------------------------------------------------------
Procedure AddOneVar(SName,SType,STranscient:string);
Var Path : String;
Begin
  XML_Write(idTx,'Variable-SD/+Variable','name',SName);
  Path := 'Variable-SD/#'+IntToStr(NodeNum);
  XML_Write(idTx,Path,'Type',SType);
  XML_Write(idTx,Path,'Transcient',STranscient);
  NodeNum := NodeNum + 1;
end;
//----------------------------------------------------------------------
Procedure SendVariablesList;
Begin
  idTx := XML_Create;
  NodeNum := 0;
  AddOneVar('Clock','TimeStamp','false');
  AddOneVar('Var2','float','false');
  AddOneVar('Var3','int','true');
  XML_Answer(idTx,'@');
End;
//----------------------------------------------------------------------
Procedure AddOneAlarm(AName,SDescription,VName:string);
Var Path : String;
Begin
  XML_Write(idTx,'Alarm-SD/+Alarm','name',AName);
  Path := 'Alarm-SD/#'+IntToStr(NodeNum);
  XML_Write(idTx,Path+'/Description','',SDescription);
  if length(VName)>0 then XML_Write(idTx,Path+'/Variable','Name',VName);
  NodeNum := NodeNum + 1;
End;
//----------------------------------------------------------------------
Procedure SendAlarmsList;
Begin
  idTx := XML_Create;
  NodeNum := 0;
  AddOneAlarm('EquipmentOnFire','Equipment environment critically overheated','Temperature');
  AddOneAlarm('Motor_3_failure','Conveyor motor 3 electrical or mechanical failure.','');
  XML_Answer(idTx,'@');
End;
//----------------------------------------------------------------------
Procedure SendRecipesList;
Begin
  idTx := XML_Create;
  XML_Write(idTx,'RecipeList-Response/+Recipe','','Ma recette 1');
  XML_Write(idTx,'RecipeList-Response/+Recipe','','Ma recette 2');
  XML_Write(idTx,'RecipeList-Response/+Recipe','','Ma recette 3');
  XML_Answer(idTx,'@');
End;
//----------------------------------------------------------------------
Procedure SendPingResponse;
Begin
  idTx := XML_Create;
  MessageType := 'PING-RESPONSE';
  XML_Answer(idTx,'@');
End;
//----------------------------------------------------------------------
Begin
  idRx := XML_Create;
  if XML_ReceiveBuf(idRx) then
  Begin
    if compareText('DATA-REQUEST',MessageType)=0 then
    Begin
      S := XML_Read(idRx, '', '','?');
      if (CompareText('Variable-SD-Query',S)=0)  then SendVariablesList;
      if (CompareText('Alarm-SD-Query',S)=0)     then SendAlarmsList;
      if (CompareText('RecipeList-Request',S)=0) then SendRecipesList;
    end;
    if compareText('PING-REQUEST',MessageType)=0 then SendPingResponse;
  End;
End;
```

See also: XML functions, ASCII server module

XML_Send

(USED BY ASCII regulator/device ONLY)

Sends an XML document to a server

Format

Function XML_Send(IdXML:integer;header:String;WaitAnswer:Boolean):Boolean;

Parameters

- **IdXML**: XML document identifier (see XML_Create and XML_CreateGlobal function)
- **Header**: String to add before the XML document
- **WaitAnswer**: when true, the script will wait for an answer from the server before to continu execution.

Returned value

Returns true if successfully completed, otherwise it returns false (after a time out when the WaitAnswer is set)

Description

This function sends an XML document to the server.

Example

```
// This is the script OnWrite of the tag identified by IdTag
Var
  S,AlarmName,Header,Severity,IsSet, SAnswer : string;
  Idx : integer;
Begin
  idx := XML_Create;

  if App.Var.Alarm.Severity=0 then Severity := 'low';
  if App.Var.Alarm.Severity=1 then Severity := 'warning';
  if App.Var.Alarm.Severity=2 then Severity := 'critical';

  if App.Var.Alarm.IsSet then IsSet := 'true'; else IsSet:='false';

  Header := 'XPTP/1.0 NOTIFY<cr><lf>Content-Length: <len><cr><lf>Content-type: text/xml; utf-
8<cr><lf>Message-ID:'+IntToStr(App.Var.Message_Id)+'<cr><lf><cr><lf><?xml version="1.0"
encoding="utf-8"?><cr><lf>';

  XML_Write(idx,'Alarm','Name',App.Var.Alarm.Name);
  XML_Write(idx,'Alarm','Time',FormatDateTime('YYYY-MM-DD"T"HH:MM:SS',GetNow));
  XML_Write(idx,'Alarm','Severity',Severity);
  XML_Write(idx,'Alarm','IsSet',IsSet);
  XML_Write(idx,'Alarm/V','name',App.Var.Alarm.VName);
  XML_Write(idx,'Alarm/V','',App.Var.Alarm.Vtext);


  App.Var.Message_Id := App.Var.Message_Id+1;
  if App.Var.Message_Id>$FFFFFFFF then App.Var.Message_Id:=0;

  SAnswer := XML_Send(idx,Header,true);
  WriteConsole('Answer= '+SAnswer);

  WriteTagValue(IdTag,0,false);
End;
```

See also: XML functions, ASCII server module