TMS TAdvSpreadGrid
DEVELOPERS GUIDE

Documentation: Nov, 2016

Web: http://www.tmssoftware.com
Email: info@tmssoftware.com

Table of contents

Availability

TMS TAdvSpreadGrid is available as VCL component set for Win32/Win64 application development.

TMS TAdvSpreadGrid is available for Delphi 7, 2007, 2009, 2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, 10 Seattle, 10.1 Berlin, C++Builder 2007, 2009, 2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, 10 Seattle, 10.1 Berlin (Prof/Enterprise/Architect)

TMS Grid Pack has been designed for and tested with: Windows 2000, 2003, 2008, XP, Vista, Windows 7, Windows 8, Windows 10.

Online references

TMS software website:
http://www.tmssoftware.com

TMS TAdvSpreadGrid page:
http://www.tmssoftware.com/site/aspgrid2.asp

Description

Powerful spreadsheet function calculation support added to the full TAdvStringGrid feature set.

	A	B	C	D	
1	1	=SUM(A1:A10)	1	=C1+C2	
2	2	=AVERAGE(A1:A10)	2	=C1-C2	
3	3	=COUNT(A1:A10)	3	=C1*C2	
4	4	=STDEV(A1:A10)	4	=C1/C2	
5	5	=VAR(A1:A10)		=D1+D2+D3+D4	
6	6	=MAX(A1:A10)			
7	7	=MIN(A1:A10)			
8	8	=DEVSQ(A1:A10)	0.5	=SIN(C8)	sir
9	9			=COS(C8)	co
10	10			=TAN(C8)	ta
11				=COTAN(C8)	co

Features

TAdvSpreadGrid extends the full power of TAdvStringGrid with formulas.

- Simple formula editing interface
- Auto recalculation
- Native XLS file import and export*
- Single cell recalculation, full recalculation
- Extensive range of mathematical functions
- Save with formulas or formula results only
- Single cell references in formulas
- Cell range formulas
- Formula precision for grid on cell basis
- Display formulas or formula results
- Date / time functions
- Intelligent formula aware copy and paste
- Can be extended with custom functions
- Cell names
- Cell name mode can be RxCy style or A1-style
- Can reference cells from other TAdvSpreadGrids
- Math library infrastructure to allow easy extending
- TAdvSpreadGrid with custom functions
- Includes the free ESBMaths library with 46 scientific constants and 19 mathematical functions from ESB Consultancy
- ESBPCS maths library available for even more statistical functions
- Intelligent and customizable hints while editing formulas

Cell acces and function overview

This is an overiew of using cell references and built-in functions in TAdvSpreadGrid. Formulas can contain cell references, constants, single parameter functions, multi parameters functions, cell name references and cell range functions.

Cell references

If CellNameMode is nmRC then cell references are in RxCx format, where x is the row number and y is the column number. If CellNameMode is nmA1 then a cell reference consists of 2 parts : the column identifier and the row identifier. The column identifier is a character, starting from A for the first column, B for the second column, etc.. After column 26, the column identifier is a double character string AA, AB, etc... The row identifier starts at 1 for the first editable row.

Example

cell 1,1 is A1, cell 2,2 is B2, cell 27,27 is AA27

Cell ranges are specified by the top left cell and bottom right cell. As such, the first 15 cells in column 1, can be specified as A1:A15.

Cell formulas are by default relative. That means that when cell formulas are involved in copy & paste operations or row/column insert and delete, TAdvSpreadGrid will automatically adapt the formulas to address the proper relative cells. Absolute cell addresses will not be modified during clipboard copy & paste operations or during row/column insert and delete. Prefix the cell address row or column part with '$' to indicate an absolute cell address.

Example

A$1 : A is a relative column address, 1 is an absolute row address

$B$2 : B is an absolute column address, 2 is an absolute row address

Cell ranges

Cell ranges are identified by topleft cell & bottomright cell split by ':'

Example

A1:B3 : specifies the range of cells from cell 1,1 to cell 2,3

$A$1:$B$3 : specifies an absolute cell range from cell 1,1 to cell 2,3

Constants

PI, E,True,False

Single parameter functions

ABS(parameter) : absolute value

ROUND(parameter) : rounds value

TRUNC(parameter) : truncates value

CEILING(parameter; significance) : rounds the parameter to the nearest multiple of significance

FRAC(parameter) : returns fractional part of value

FACT(parameter) : factioral of value

INT(parameter) : int part of value

SIN(parameter) : sine of value

COS(parameter) : cosine of value

TAN(parameter) : tangens of value

COTAN(parameter) : cotangens of value

SINH(parameter) : hyperbolic sine of value

COSH(parameter) : hyperbolic cosine of value

TANH(parameter) : hyperbolic tangens of value

COTANH(parameter) : hyperbolic cotangens of value

ASIN(parameter) : arcsin of value

ACOS(parameter) : arccos of value

ATAN(parameter) : arctangens of value

ACOTAN(parameter) : arccotangens of value

LN(parameter) : natural logarithm of value

LOG2(parameter) : base 2 logarithm of value

LOG10(parameter) : base 10 logarithm of value

EXP(parameter) : exponential of value

RAND(parameter) : random between 0 and value

RADIANS(parameter) : converts degrees to radians

DEGREES(parameter) : converts radians to degrees

SQR(parameter) : square of value

SQRT(parameter) : square root of value

CUBE(parameter) : cubic square of value

CHS(parameter) : change sign

POWER(parameter,exp) : parameter to exponent exp

Multi parameter functions :

LT(param1;param2) : larger than : returns 1 of param1>param2

ST(param1;param2) : smaller than : returns 1 of param1<param2

EQ(param1;param2) : equal : returns 1 of param1=param2 else 0

CHOOSE(sel;param1;param2) : returns param1 if sel>0 else param2

Cell range functions

SUM(range) : sum of all cell values in range

PRODUCT(rangfe) : product of all cell values in range

AVERAGE(range) : average of all cell values in range

MIN(range) : min. cell value in range

MAX(range) : max. cell value in range

COUNT(range) : nr. of cells in range

COUNTA(range) : nr of non blank cells in range

COUNTIF(range;condition) : nr of cells meeting condition in range

STDEV(range) : standard deviation of range

STDEVP(range) : standard deviation of total population of range

DEVSQ(range) : sum of squares of deviations of range

VAR(range) : variance of range

Date & Time functions

HOUR(parameter) : gets the hour from a cell containing a valid time string

MIN(parameter) : gets the minute from a cell containing a valid time string

SECOND(parameter) : gets the second from a cell containing a valid time string

DAY(parameter) : gets the day from a cell containing a valid time string

MONTH(parameter) : gets the month from a cell containing a valid time string

YEAR(parameter) : gets the year from a cell containing a valid time string

WEEKDAY(parameter) : gets the day of the week from a cell containing a valid time string

TODAY : gets the current day

NOW : gets the current time

Logical functions

AND(parameters) : logical AND function

OR((parameters) : logical OR function

NAND(parameters) : logical NAND function

NOR((parameters) : logical NOR function

XOR((parameters) : logical XOR function

NOT(parameter) : logical NOT function

TRUE : constant returning true

FALSE: constant returning false

String functions

LEN(parameter) : returns the length of a string value

LOWER(parameter) : returns string in lowercase

UPPER(parameter) : returns string in uppercase

CONCATENATE(parameter list) : returns concatenated string of parameters

SUBSTITUTE(param text; param oldtext; param new text) : returns string with oldtext replaced by newtext

LEFT(param string;len integer) : returns first len charactares of string

RIGHT(param string;len integer) : returns last len characters of string

MID(param string; pos; len: integer): returns len characters starting from position pos in string

TRIM(param) : removes all spaces from text except spaces between words

SEARCH(find text; text) : returns position of string find text in text

LOOKUP(param; range1, range2) : returns the value of the element in range2 that has the index of the matching element in range1 for param

MATCH(lookup; range) : returns the index of the element param in the range

INDEX(range; val1, val2) : returns the value of element at index val1,val2 in the range

Custom function libraries

TAdvSpreadGrid allows to use libraries that extend the built-in functions. A function library is a class that descends from TMathLib defined in the unit AdvPars. In order to be able to use multiple different function libraries simultaneously, a TLibBinder component can be assigned to TAdvSpreadGrid. Multiple TMathLib components can be assigned to the TLibBinder.

Anatomy of TMathLib

TMathLib implements a number of public virtual functions that can be overridden to implement custom functions:

```
function HandlesFunction(FuncName:string):Boolean;

function HandlesStrFunction(FuncName:string):Boolean;

function CalcFunction(FuncName:string;Params:TParamList;var
ErrType,ErrParam: Integer):Double;

function CalcStrFunction(FuncName:string;Params:TStringList;var
ErrType,ErrParam: Integer):string;

function GetEditHint(FuncName:string;ParamIndex: Integer):string;
```

Implemented functions

The methods HandlesFunction and HandlesStrFunction are simple methods being called by the TLibBinder and assumed to return true when the library implements the function with the name 'FuncName'.

Example

```
function TMiscMathLib.HandlesFunction(FuncName: string): boolean;

begin

  Result := (FuncName = 'HARMEAN') or

    (FuncName = 'GEOMEAN');

end;
```

This shows a library implementing 2 statistical functions HARMEAN and GEOMEAN. HandlesFunction should return true for functions that return a floating point result type. If a library implements a function with a string result, the HandlesStrFunction should be used.

Function calculation

The method CalcFunction implements the actual calculation of the function. The first parameter is the function name that should be calculated, the second parameter is a list of function parameters. The 2 last parameters can be set if an incorrect parameter is specified and the index of this incorrect parameter.

Example

```
function TMiscMathLib.CalcFunction(FuncName: string; Params: TParamList;
   var ErrType,ErrParam: Integer): Double;
var
   k: Integer;
   d: Double;

begin
   Result := 0.0;

   ErrType := Error_NoError;

   if FuncName = 'HARMEAN' then
   begin
       d := 0;
       for k := 1 to Params.Count do
       begin
           if Params.Items[k - 1] <> 0 then
           begin
               d := d + (1 / Params.Items[k - 1]);
           end
           else
           begin
               ErrType := Error_DivisionByZero;
               ErrParam := k - 1;
           end;
         Result := 1/d * Params.Count;
       end;
      Exit;
   end;

   if FuncName = 'GEOMEAN' then
   begin
       d := 1;
       for k := 1 to Params.Count do
       begin
           d := d * Params.Items[k - 1];
       end;
       if Params.Count > 0 then
       begin
           Result := exp(1/Params.Count * ln(d));
       end
       else
       begin
           ErrType := Error_DivisionByZero;
           ErrParam := 0;
       end;
   end;
end;
```

In this example, the functions HARMEAN and GEOMEAN are implemented. This shows how the method loops through the number of parameters passed to the function and calculates the result. As shown here in the code, a parameter of these functions cannot be zero. If a zero parameter is found, the method sets the error type to Error_DivisionByZero and if needed the index of the parameter that caused the actual error. The supported types of errors are:

Error_NoError = 0;

Error_NoFormula = 1;

Error_DivisionByZero = 2;

Error_InvalidValue = 3;

Error_InvalidCellRef = 4;

Error_InvalidRangeRef = 5;

Error_InvalidGridRef = 6;

Error_InvalidNrOfParams = 7;

Error_CircularReference = 8;

Error_NoOpenParenthesis = 9;

Error_NoCloseParenthesis = 10;

Error_PrematureEndOfFormula = 11;

Error_UnknownError = 12;

Error_InvalidQualifier = 13;

Error_InvalidTokenAtPosition= 14;

Error_Overflow = 15;

Error_Underflow = 16;

Error_CircularRange = 17;

The string function handling is equivalent, as shown in this sample code:

```
function TStringMathLib.CalcStrFunction(FuncName: string;
   Params: TStringList; var ErrType, ErrParam: Integer): String;
var
   i: Integer;
   s: string;
begin
   Result := '';
   ErrType := 0;

   if Params.Count = 0 then
   begin
       ErrType := Error_InvalidNrOfParams;
       Exit;
   end;

   Result := '';

   if FuncName = 'REVERSE' then
   begin
```

```
      s := Params.Strings[0];
      for i := 1 to Length(s) do
        Result := Result + s[Length(s)- i + 1];
  end;

  if FuncName = 'CAPITALIZE' then
  begin
      s := Params.Strings[0];
      for i := 1 to Length(s) do
        if (i = 1) or ( (i > 1) and (s[i - 1]=' ')) then
            Result := Result + upcase(s[i])
        else
          Result := Result + s[i];
  end;

end;

function TStringMathLib.HandlesStrFunction(FuncName: string): Boolean;
begin
  Result := (FuncName = 'REVERSE') or
            (FuncName = 'CAPITALIZE');
end;
```

Function parameter hints

An additional feature of a function library is the capability to provide parameter hints while entering the function in TAdvSpreadGrid. This capability is enabled with the method : GetEditHint. This method is called during editing when entering parameters for a function that the library implements. The library should return a string that contains the hint text. This hint text can contain HTML formatting tags. The ParamIndex parameter of the method GetEditHint indicates the index of the parameter being entered. This can be used to highlight with a tag, the parameter being entered.

Example

```
function TMyMathLib.GetEditHint(FuncName: string;
  ParamIndex: Integer): string;
begin
  if FuncName = 'MYPROC' then
  begin
    case ParamIndex of
      1: Result := 'MyProc(<b>n: Integer Value: string)<HR>Returns the
n''character from string Value';
      2: Result := 'MyProc(n: Integer;<b> Value: string)<HR>Returns the
n''character from string Value';
      end;
  end;
end;
```

In this example, the first parameter is bold, when the user is entering the first parameter and the 2nd parameter is bold when the user is entering the 2nd parameter.

Tips and FAQ

Using custom functions directly

The OnIsCustomFunction and OnCalcCustomFunction events can be used to implement custom functions for TAdvSpreadGrid. Via the event OnIsCustomFunction event, the grid is informed what names should be treated as custom functions and the OnCalcCustomFunction performs the actual calculation of the function. With the example code presented, the MYTEST function is added that returns as a result parameter * 2:

```
procedure TForm2.AdvSpreadGrid1CalcCustomFunction(sender: TObject;
  var func: string; var param: Double);
begin
  if func='MYTEST' then
    param := param *2;
end;

procedure TForm2.AdvSpreadGrid1IsCustomFunction(sender: TObject;
  var func: string; var match: Boolean);
begin
  match := func='MYTEST';
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
  advspreadgrid1.Cells[1,1] := '1';
  advspreadgrid1.Cells[1,2] := '=MYTEST(A1)';
  advspreadgrid1.Recalc;
end;
```

When try to install TAdvSpreadGrid, I get an error 'AdvSprd.pas(187): E2003 Undeclared identifier: "TVAlignment ... Could not compile used unit "AdvSprd.pas" ""

Add the unit AdvObj to the uses clause of AdvSprd.pas and this issue should be fixed.

How to use the lookup function

LOOKUP(param; range1, range2) : returns the value of the element in range2 that has the index of the matching element in range1 for param

Example:

This code snippet shows how the lookup is performed for the value of 2 in range A1:A5 and the value is retrieved from range B1:B5

```
begin
  advspreadGrid1.Cells[1,1] := '1';
```

```
  advspreadGrid1.Cells[1,2] := '2';
  advspreadGrid1.Cells[1,3] := '3';
  advspreadGrid1.Cells[1,4] := '4';
  advspreadGrid1.Cells[1,5] := '5';

  advspreadGrid1.Cells[2,1] := 'BMW';
  advspreadGrid1.Cells[2,2] := 'Audi';
  advspreadGrid1.Cells[2,3] := 'Ferrari';
  advspreadGrid1.Cells[2,4] := 'Mercedes';
  advspreadGrid1.Cells[2,5] := 'Porsche';

  advspreadGrid1.Cells[3,1] := '=LOOKUP("2";A1:A5;B1:B5)';
  advspreadgrid1.Recalc;
end;
```

How to add formula support in TAdvSpreadGrid to access DB tables

With this small DB sample mathlib, the concept is demonstrated to add formula support in TAdvSpreadGrid to access DB tables:

```
type

  TDBMathLib = class(TMathLib)
  private
    { Private declarations }
    FDataSource: TDataSource;
  protected
    { Protected declarations }
    procedure Notification(AComponent: TComponent; AOperation: TOperation); override;
  public
    { Public declarations }
    function HandlesStrFunction(FuncName:string):Boolean; override;
    function CalcStrFunction(FuncName:string;Params:TStringList;var ErrType,ErrParam:
Integer):string; override;
  published
    { Published declarations }
    property DataSource: TDataSource read FDataSource write FDataSource;
  end;


implementation

{ TDBMathLib }

function TDBMathLib.CalcStrFunction(FuncName: string; Params: TStringList;
  var ErrType, ErrParam: Integer): string;
var
  s: string;
  n,e: integer;
```

```
  fld: TField;
begin
  if (FuncName = "DBV") then
  begin
    if Params.Count <> 2 then
    begin
      ErrType := Error_InvalidNrOfParams;
      Exit;
    end;

    if not Assigned(DataSource) then
    begin
      ErrType := Error_NoDataSource;
      Exit;
    end;

    if not Assigned(DataSource.DataSet) then
    begin
      ErrType := Error_NoDataSet;
      Exit;
    end;

    if not DataSource.DataSet.Active then
    begin
      ErrType := Error_NoDataSetActive;
      Exit;
    end;

    s := Params.Strings[0]; // DB FIELD value

    fld := DataSource.DataSet.FieldByName(s);

    if not Assigned(fld) then
    begin
      ErrType := Error_InvalidValue;
      ErrParam := 1;
    end
    else
    begin
      val(Params.Strings[1],n, e);

      DataSource.DataSet.First;
      DataSource.DataSet.MoveBy(n);

      Result := fld.AsString;
    end;
  end;
end;

function TDBMathLib.HandlesStrFunction(FuncName: string): Boolean;
begin
```

```
  Result := FuncName = "DBV";
end;

procedure TDBMathLib.Notification(AComponent: TComponent;
  AOperation: TOperation);
begin
  inherited;
  if (AOperation = opRemove) and (AComponent = FDataSource) then
    FDataSource := nil;
end;
```

How to show negative values in red within a cell with a formula

This code snippet shows how to have negative values in red within a cell with a formula:

```
procedure TForm1.AdvSpreadGrid1GetCellColor(Sender: TObject; ARow,
  ACol: Integer; AState: TGridDrawState; ABrush: TBrush; AFont: TFont);
begin
   if AdvSpreadGrid1.CalculatedValue[acol,arow] < 0 then
    AFont.Color := clred;
end;
```