TSpread Component

TSpread is an enhanced StringGrid which supports individual cell fonts, display formats, background colors, bitmaps, cell frames and cell arithmetic. In addition, TSpread supports buttons, check boxes, combo boxes and spinedits in cells. The display attributes of each individual cell can be controlled by a call to one of the methods provided by the component. Mathematical formulas can be entered directly into a cell by the user or set programmatically in your code.

Read This Before You Start !
Properties
Methods
Events
Cell Arithmetic
Adding components to the grid

Other Topics:

What's new in version 2.0 ?
Registration
Support
Revision History
Can I get the Source Code ?

Before You Start

Important Notes:

Before using the TSpread component review the following topics for important information.

Initialize Method
AddRow Method
AddCol Method

Initialize Method

Important Note:

This page has been left in for documentation purposes. If you have version 2.5 of Tspread or later, you no longer have to call the initialize method in your code before using the component.

Call the initialize method once in your application before performing any operations on the Tspread component. The initialize method sets up internal object tables used for bitmaps, fonts, brushes and other cell objects. The formcreate event would be a good place to put the call.

Example:

```
procedure Tform1.formcreate(Sender:TObject);
begin
        Spread1.initialize;
end;
```

If you call this method more than once, it will be ignored.

See Also:

Before You Start!

AddRow Method

With the standard string grid you simply add rows to the grid by changing the value of the rowcount property. TSpread will allow you to do this as well, but because of the internal housekeeping TSpread must do you should use the AddRow method instead. If you change the rowcount property any objects you have in the grid (such as bitmaps, combos, etc.) cannot be properly maintained.

Syntax:

Spread1.Addrow;

adds a row to the end of the spreadsheet.

In other words, do not use the Rowcount property, Use AddRow instead to add rows.

AddCol Method

With the standard string grid you simply add columns to the grid by changing the value of the colcount property. TSpread will allow you to do this as well, but because of the internal housekeeping TSpread must do you should use the AddCol method instead. If you change the colcount property any objects you have in the grid (such as bitmaps, combos, etc.) cannot be properly maintained.

Syntax:

Spread1.Addcol;

adds a row to the end of the spreadsheet.

In other words, don't use the colcount property, use the AddCol method instead to add cols.

Properties

CalcOnChange
CalcOrder
Decimals
DefaultCellAlignment
DefaultCellFormat
DefaultCellType
ShowOptions
ShowZeros
StretchBitmap

CalcOnChange Property

CalcOnChange is a boolean property which determines if the grid is automatically recalculated when a cell's contents are changed.

See Also:

Recalculate Method
CalcOrder Property

Recalculate Method

The recalculate method causes all cells in the grid to be recalculated.

Syntax: Spread1.Recalculate; {recalculate the grid}

See Also:

CalcOnChange Property
CalcOrder Property

CalcOrder Property

The CalcOrder property determines the order in which recalculations are made. This property can have one of the following values:

- coCols - Calculations are made in column order, i.e. the first column of the grid is calculated for each row and then the next column for each row and so forth until all columns have been calculated.
- coRows - Calculations are made in row order, i. e. the first row of the grid is calculated for each column and then the next row for each column and so forth until all rows have been calculated.

See Also:

CalcOnChange Property
Recalculate Method

Decimals Property

Decimals is an integer property which determines the default number of decimals to display for computed values in a cell.

DefaultCellAlignment Property

DefaultCellAlignment determines the default alignment for every cell in the grid. This property can take one of the following values:

- taLeftJustify - left justify the cell contents.
- taRightJustify - right justify the cell contents.
- taCenter - center the cell contents in the cell.

DefaultCellFormat Property

The DefaultCellFormat property can take one of the following values:

- ffFixed - Floating point fixed format.
- ffCurrency - Floating point currency format.
- ffNumber - Floating point number format.
- ffGeneral - Floating point general format.

Note:

These are the same as described in the Delphi help file for the Floattostrf procedure.

DefaultCellType Property

The DefaultCellType property determines the initial cell type of all cells in the grid. When the initialize method is called, each cell is set to the DefaultCelltype.

See Also:

GetCellType Method
SetCellType Method
Initialize Method
What are Cell Types ?

GetCellType Method

Use GetCelltype to determine the type of a particular cell.

Example:

```
var
        T : Tctype;
begin
        T := spread1.GetCellType(9,8);
end;
```

Gets the celltype for the cell at column 9 and row 8.

See Also:

Cell types
Using Cell types
SetCellType Method

Cell Types

The concept of cell types has been introduced with version 2.0 of TSpread.
A cell type may be one of the following:

1 ctnone - No specified cell type
2 ctformula - A cell by default will contain a formula
3 ctdate - A cell by default will contain a date
4 cttime - A cell by default will contain a time
5 cttext - A cell by default will contain text
6 ctobject - A cell will contain an object such as a button or check box.

Note:

When putting objects in cells it is not necessary to call SetCelltype to set the cell to ctobject. The component does this automatically.

See Also:

Do I have to use cell types ?

Using Cell Types

It can be useful but not mandatory that you use cell types. If you want to restrict, for example, entries in column 1 to dates only, you could use the setcelltype to set the celltype of each cell to ctdate. If the application user then entered something in the cell which could not be interpreted as a valid date, then the oncelltypeviol event is fired and you can display a message box or take whatever corrective action is appropriate under the circumstances.

If you don't want to use cell types just set the DefaultCellType property to ctnone. In the OnCelltypeViol event handler you'll only need one line of code:

changecelltype := true;

This will allow the cell type to be changed to whatever type of data was entered. If an entry can be interpreted as a formula then it will and it will be calculated displaying the results, else it will be interpreted as text.

See Also:

OnCelltypeViol Event

OnCellTypeViol Event

The OnCellTypeViol event is fired when data which is inconsistent with the cell type is entered into a cell. The declaration looks like this:

```
procedure TForm1.Spread1CellTypeViol(icol, irow: Longint; Celltype,
  CellEntrytype: Tctype; cellstring: String; var NewCellstring: OpenString;
  var changecelltype: Boolean);
```

icol and irow are the column and row of the offending cell.
celltype is the cell type of the cell.
entrytype is the type of entry made in the cell the values may be the same as used for cell types except for ctobject.
cellstring is the contents of the cell.
newcellstring is an empty string which can be set with a value such as 'ERR' that is to be displayed in the cell on return from the handler.
changecelltype is initially false, If you want to allow the type to be changed then set changecelltype to true before the handler exits.

Note:

If newstring is set to something other than an empty string on return from your handler code, the OnCellEntryError event will not be fired. This means you can handle the error in either handler.

SetCellType Method

Use SetCelltype to set the cell type for a particular cell.

Example:

SetCelltype(1,1,ctdate); {sets cell at col 1 and row 1 to ctdate}

See Also:

Cell Types
Using Cell types
GetCellType Method

ShowOptions Property

The setting of the ShowOptions property determines what is displayed in a cell when it obtains the focus. This property can have one of two values:

- ssformulas - Formulas are displayed when the cell obtains the focus.
- ssvalue - computed values are displayed when the cell obtains the focus.

ShowZeros Property

ShowZeros is a boolean property that determines if a zero value should be displayed when a cell value or computed value results in zero.

StretchBitmap Property

StetchBitmap is a boolean property that determines whether to size a bitmap to the size of the cell. When this property is true, a cell containing a bitmap is sized to the size of the bitmap, when false the cell size is unchanged.

See Also:

Putbitmap method

Putbitmap Method

Use the putbitmap method to display a bitmap in a cell. The bitmap will continue to be displayed until a call to the clearbitmap method is made.

Example:

```
procedure TForm1.button1click(Sender:TObject);
var
      T : TBitmap;
begin
      T := TBitmap.create;
      T.LoadFromfile('d:\delphi\images\splash\16color\athena.bmp');
      Spread1.putbitmap(1,2,T);
end;
```

The above example displays the athena bitmap in the cell at column 1 and row 2.

Note:

TSpread frees the memory associated with the bitmap when the application terminates so it's not necessary for you to include any code to free the object.

See Also:

ClearBitmap

Clearbitmap Method

The clearbitmap method clears a bitmap from a cell which was previously loaded with a call to putbitmap.

Example:

Spread1.clearbitmap(1,2);

The example clears the bitmap in column 1 and row 2.

See Also:

PutBitmap

Methods

Important Note:
Before any operations are performed on the TSpread component you MUST call the initialize method.

Initialize
AddRow
AddCol
Deleterow
Deletecol
GetFormula
SetFormula
Recalculate
HideRow
HideCol
UnhideRow
UnhideCol
Putbitmap
ClearBitmap
SetCellFormat
SetColFormat
SetRowFormat
GetCellFormat
GetCellFont
SetCellFont
SetColFont
SetRowFont
SetCellFrame
GetCellAlignment
SetCellAlignment
SetColAlignment
SetRowAlignment
GetCellBrush
SetCellBrush
SetColBrush
SetRowBrush
GetCellType
SetCellType
InsertRow
InsertCol
IsCellLocked
LockCell
LockCol
LockRow
LockAllCells
UnLockCell
UnLockCol
UnLockRow

UnLockAllCells
SumColumn
SumRow
LoadFromFile
SaveToFile
MakeNewSheet

See other methods under:

Adding Components to the Grid
Accessing / changing component properties

DeleteRow Method

With the standard string grid you simply delete rows from the grid by changing the value of the rowcount property. TSpread will allow you to do this as well, but because of the internal housekeeping TSpread must do you should use the DeleteRow method instead. If you change the rowcount property any objects you have in the grid (such as bitmaps, combos, etc.) cannot be properly maintained.

Syntax:

Spread1.Deleterow(20); {rownumber is a longint}

deletes row 20 from the grid.

See Also:

DeleteCol Method

DeleteCol Method

With the standard string grid you simply delete columns from the grid by changing the value of the colcount property. TSpread will allow you to do this as well, but because of the internal housekeeping TSpread must do you should use the DeleteCol method instead. If you change the colcount property any objects you have in the grid (such as bitmaps, combos, etc.) cannot be properly maintained.

Syntax:

Spread1.Deletecol(10); {colnumber is a longint}

deletes row 10 from the grid.

See Also:

DeleteRow Method

GetFormula Method

The GetFormula method is a function which returns the formula for a given cell as a string. This method takes two longint arguments for the column and row.

Example:

formula := Spread1.getformula(1,2);

The above example gets the formula associated with col 1 and row 2.

SetFormula Method

SetFormula associates a formula with a cell.

Example:

```
SetFormula(1,2,'=a1+b1');
```

The above example sets the formula for column 1 and row 2 to '=a1+b1'.

See Also:

Cell Arithmetic

Cell Arithmetic

Each TSpread cell can contain text, numeric values or formulas. Formulas can be any valid mathematical expression such as '=4^2+3'. In addition to mathematical expressions, cell references are also supported. For example if a cell contained '=a1*b3' the calculation would be the product of multiplying column 1, row 1 by column 2, row 3. Letters represent column references and numbers represent row references similar to most spreadsheet programs. The only spreadsheet functions presently supported are @sum, @date and @time. Unless a formula is a function such as @date or @time or @sum, the first character of the formula MUST BE an equal sign '='.

See Also:

@Sum function
@Date function
@Time function
Arithmetic Operators

@Sum Function

A Cell can contain the @sum function which takes two arguments separated by a period. The first argument is the starting cell reference in the form 'a1' where the letter maps to the column number and the number is the row reference. The second argument is the ending cell to include in the computation.

Example:

@sum(a1.a10)

This cell function causes the cell value to be set to the sum of col 1 row 1 through col 1 row 10.

Date Function

Dates may be entered into a cell by using the @date function. The syntax for the function is:

@date(mo,day,yr)

Example:

@date(1,31,96) {cell entry for Jan 31, 1996}

Note:

If a cell has a celltype of ctdate then the date may be entered as a string but only in short date format, such as '1/1/96' without the need for the @date function.

Time Function

Times may be entered into a cell by using the @time function. The syntax for the function is:

@time(hr,min,sec)

Uses 24 hour (military) time.

Example:

@time(14:00:00) {2:00 pm}

Note:

If a cell has a celltype of cttime then the time may be entered as a string such as '14:00:00' without the need for the @time function.

Arithmetic Operators

The following is a list of legal operators and their precedence. Operators are evaluated according to precedence. Higher precedence operators are evaluated before operators of lower precedence.

Operator	Precedence	
^	3	exponential
*	2	multiplication
/	2	division
!	2	factorial
+	1	addition
-	1	subtraction

HideRow Method

The HideRow Method causes a row to be hidden when the grid is displayed.

Example:

Spread1.HideRow(1);

The examples hides row 1 of the grid.

See Also:

UnHideRow

UnHideRow Method

UnHideRow causes a previously hidden row to be redisplayed.

Syntax:

Spread1.Unhiderow(2); {unhide row 2}

See Also:

HideRow

HideCol Method

The HideCol method causes a column to be hidden when the grid is displayed.

Example:

```
Spread1.HideCol(1);
```

The example hides column 1 of the grid.

See Also:

UnHideCol

UnHideCol Method

UnHideCol causes a previously hidden column to be redisplayed.

SetCellFormat Method

The SetCellFormat declaration looks like this:

procedure TSpread.setCellFormat(icol,irow:longint;F:TFloatFormat;Prec:integer);

Use SetCellFormat to set the format of an individual cell.

Example:

Spread1.SetCellFormat(1,2,ffCurrency,2);

The example sets the format of column 1 and row 2 to ffCurrency with two decimal places.

See Also:

SetColFormat
SetRowFormat

SetColFormat Method

The SetColFormat declaration looks like this:

procedure TSpread.setColFormat(icol:longint;F:TFloatFormat;Prec:integer);

Use SetCellFormat to set the format of all cells in a specified column.

Example:

Spread1.SetColFormat(2,ffCurrency,2);

The example sets the format of all cells in column 2 to ffCurrency with two decimal places.

SetRowFormat Method

The SetRowFormat declaration looks like this:

procedure TSpread.SetRowFormat(irow:longint;F:TFloatFormat;Prec:integer);

Use SetCellFormat to set the format of all cells in a specified column.

Example:

Spread1.SetRowFormat(2,ffCurrency,2);

The example sets the format of all cells in row 2 to ffCurrency with two decimal places.

GetCellFormat Method

The GetCellFormat Method is a function which returns a record of type TPrec.

TPrec is declared as follows:

```
TPrec = record
   Fmt : TFloatFormat;
   Dec : integer;
end;
```

Use GetCellFormat to retrieve the format and number of decimals of precision for a given cell.

Example:

```
TPrec = record
   Fmt : TFloatFormat;
   Dec : integer;
end;
...
...
...
procedure TForm1.button1click(Sender:TObject);
var
   T : TPrec;
begin
   T := GetCellFormat(1,2);
end;
```

The example gets the format for the cell at column 1 and row 2. T.Fmt will contain either ffFixed, ffNumber, ffCurrency or ffGeneral. T.Dec will contain the number of decimals of precision for the cell.

GetCellFont Method

The GetCellFont method is a function which returns a type TFont representing the current font for a given cell.

Example:

```
procedure TForm1.button1click(Sender:TObject);
var
   T : TFont;
begin
      T := Spread1.GetCellFont(1,2);
end;
```

The example returns the current font for the cell at column 1 and row 2.

SetCellFont Method

Use the SetCellFont method to set the font for an individual cell.

Example:

```
procedure TForm1.button1click(Sender:TObject);
begin
      If FontDialog1.execute then
       Spread1.SetCellFont(1,2,FontDialog1.Font);
end;
```

The example sets the font for the cell at column 1 and row 2 to the font selected by the user from the font dialog.

See Also:

SetColFont
SetRowFont

SetColFont Method

Use the SetColFont method to set the font for all cells in a specified column.

Example:

```
procedure TForm1.button1click(Sender:TObject);
begin
      If FontDialog1.execute then
       Spread1.SetColFont(1,FontDialog1.Font);
end;
```

The example sets the font for all cells in column 1 to the font selected by the user from the font dialog.

SetRowFont Method

Use the SetRowFont method to set the font for all cells in a specified row.

Example:

```
procedure TForm1.button1click(Sender:TObject);
begin
      If FontDialog1.execute then
       Spread1.SetRowFont(1,FontDialog1.Font);
end;
```

The example sets the font for all cells in row 1 to the font selected by the user from the font dialog.

SetCellFrame Method

The setcellframe method is useful where you want to want certain cells to display a cell frame. The method declaration looks like this:

procedure TSpread.SetCellFrame(icol,irow:longint;TF:Tcframe;clr:TColor;w:integer;style:TPenStyle);

icol and irow indicate which cell to set the frame for, TF is of Tcframe which is defined below.
Clr is the pen color to use when drawing the frame, w is the width in pixels of the frame and style is the pen style to use. Refer to the Delphi help file for the TPenStyle options.

Tcframe is defined as:

Tcframe = set of (cfoutline,cfleft,cftop,cfright,cfbottom,cfnone);

Example:

```
procedure TForm1.button1click(Sender:TObject);
var
    T : Tcframe;
begin
    T := [cfleft,cfbottom,cfright];
    With Spread1 do
        Setcellframe(col,row,T,clred,1,psSolid);

end;
```

The example draws a red solid frame 1 pixel in width on the left, bottom and right of the cell at col and row.

Note:

Custom cell frames are only displayed if the cell does not currently have the focus. 2 is the maximium width you can use for the Pen, a value higher than that is ignored and 2 is assumed.

GetCellAlignment Method

The GetCellAlignment Method returns the current alignment for the given cell. The return value is of type tAlignment which will have one of the following values:

- taLeftJustify
- taRightJustify
- taCenter

Example:

```
procedure TForm1.button1click(Sender:TObject);
var
      t : tAlignment;
begin
      t := Spread1.GetCellAlignment(1,2);
end;
```

The example retrieves the current alignment setting for cell at column 1 and row 2.

SetCellAlignment Method

Use the SetCellAlignment method to set the alignment of an individual cell.

Example:

```
procedure TForm1.button1click(Sender:TObject);
begin
      Spread1.SetCellAlignment(1,2,taRightJustify);
end;
```

In the example, 1 and 2 are the column and row reference of the cell.
The third parameter is of type TAlignment.

See Also:

DefaultCellAlignment
SetColAlignment
SetRowAlignment

SetColAlignment Method

Use the SetColAlignment method to set the alignment of all cells in a specified column.

Example:

```
procedure TForm1.button1click(Sender:TObject);
begin
      Spread1.SetColAlignment(2,taRightJustify);
end;
```

SetRowAlignment Method

Use the SetRowAlignment method to set the alignment of all cells in a specified row.

Example:

```
procedure TForm1.button1click(Sender:TObject);
begin
      Spread1.SetRowAlignment(2,taRightJustify);
end;
```

GetCellBrush Method

The GetCellBrush method is a function which returns the current brush for an individual cell. The return type is TBrush.

Example:

```
procedure TForm1.button1click(Sender:TObject);
var
     Br : TBrush;
begin
      Br := TBrush.create;
      Br := Spread1.GetCellBrush(1,2);
end;
```

The example gets the current brush for the cell at column 1 and row 2.

SetCellBrush Method

Use the SetCellBrush method to set the brush for an individual cell.

Example:

```
procedure TForm1.button1click(Sender:TObject);
var
    Br : TBrush;
begin
    Br := TBrush.Create;
    Br.color := clgreen;
    Spread1.SetCellBrush(1,2,Br);
end;
```

Sets the background color of column 1 and row 2 to clgreen;

See Also:

SetColBrush
SetRowBrush

SetColBrush Method

Use the SetColBrush method to set the brush for all cells in a specified column.

Example:

```
procedure TForm1.button1click(Sender:TObject);
var
    Br : TBrush;
begin
    Br := TBrush.Create;
    Br.color := clgreen;
    Spread1.SetColBrush(1,Br);
end;
```

Sets the background color of all cells in column 1 to clgreen;

SetRowBrush Method

Use the SetRowBrush method to set the brush for all cells in a specified row.

Example:

```
procedure TForm1.button1click(Sender:TObject);
var
    Br : TBrush;
begin
    Br := TBrush.Create;
    Br.color := clgreen;
    Spread1.SetRowBrush(1,Br);
end;
```

Sets the background color of all cells in row 1 to clgreen;

InsertRow Method

Use the InsertRow method to insert a row at a specified location.

Example:

```
Spread1.InsertRow(1);
```

Inserts a new row at row 1.

InsertCol Method

Use the InsertCol method to insert a new column at a specified location.

Example:

```
Spread1.InsertCol(1);
```

Inserts a new column at column 1.

IsCellLocked Method

The IsCellLocked method is a function which returns a boolean indicating whether a particular cell is locked or not.

Example:

```
procedure Tform1.buttonclick(Sender:TObject);
var
    islocked : boolean;
begin
      With Spread1 do
      begin
          islocked := IsCelllocked(col,row);
      end;
end;
```

determines if the current cell is locked.

LockCell Method

Use the LockCell Method to prevent the user from accessing an individual cell in the grid.

The syntax is:

Spread1.lockcell(icol,irow);

icol and irow are the col and row coordinates of the cell to lock.

Note:

If icol and irow are the currently selected column and row in the grid, the lock will not take effect until the current col and row are changed either by the user or in your code.

The SelectCell event handler of TSpread will not be fired for a cell which has been locked.

See Also:

LockCol
LockRow

LockCol Method

Use the LockCol Method to prevent the user from accessing all cells in the specified column.

The syntax is:

Spread1.lockcol(icol);

See Also:

LockCell
LockRow

LockRow Method

Use the LockRow Method to prevent the user from accessing all cells in the specified row.

The syntax is:

Spread1.lockrowl(irow);

See Also:

LockCell
LockCol

LockAllCells Method

Use the LockAllCells method to lock all cells in the grid. This method takes no arguments.

Syntax:

Spread1.LockAllCells;

See Also:

UnlockAllCells

UnLockAllCells Method

Use the UnLockAllCells method to unlock all locked cells in the grid. This method takes no arguments.

Syntax:

Spread1.UnLockAllCells;

See Also:

LockAllCells

UnLockCell Method

Use the Unlockcell method to unlock a previously locked cell.

Example:

spread1.unlockcell(1,1);

{unlocks the cell at col 1 and row 1

UnLockCol Method

Use the UnLockCol method to unlock all cells in a specified column.

Syntax:

Spread1.UnLockCol(1); {unlocks column 1}

See Also:

UnLockRow

UnLockRow Method

Use the UnLockRow method to unlock all cells in a specified row.

Syntax:

Spread1.UnLockRow(3); {unlocks row 3}

See Also:

UnLockCol

SumColumn Method

The SumColumn method is a function which returns the sum of a column of values. The return type is a real.

Example:

```
r := Spread1.sumcolumn(1,2,10);
```

The example obtains the sum of column 1 starting at row 2 and ending at row 10.

SumRow Method

The SumRow method is a function which returns the sum of a row of values. The return type is a real.

Example:

```
r := Spread1.sumrow(1,2,10);
```

The example obtains the sum of row 1 starting at column 2 and ending at column 10.

LoadFromFile Method

The LoadFromFile method is new in version 2.5 of TSpread. The syntax is:

Spread1.Loadfromfile(filename:string);

The indicated file MUST be a file that was previously saved by TSpread with a call to the SaveToFile method. The file save file format is not a text file.

See Also:

SaveToFile Method
Save File Format

SaveToFile Method

The SaveToFile method is new in version 2.5 of TSpread. The syntax is:

Spread1.SaveToFile(filename:string);

This method saves the entire spreadsheet including the special cell characteristics such as brushes, fonts, buttons, comboboxes, etc. The only exceptions are - bitmaps are not saved to the file and the items property of any comboboxes are not saved.

See Also:

Save File Format
LoadFromFile Method

Save File Format

The record layout for the save file is as follows:

```
SaveRecord = record
  sCellcontents : string;
  sCellType : Tctype;
  sCellFormula : string;
  sCellFrame : Tcframe;
  sCellFramecolor : TColor;
  sCellFramewidth : integer;
  sCellFrameStyle : TPenStyle;
  sCellCol : longint;
  sCellRow : longint;
  sCellBrushcolor : TColor;
  sCellAlignment : TAlignment;
  sCellFontName : string;
  sCellFontSize : integer;
  sCellFontHeight : integer;
  sCellFontStyle : TFontStyles;
  sCellLocked : boolean;
  sCellFormat : TFloatFormat;
  sCellPrecision : integer;
  sCellBitmapname : string;
  sCellCheckBox : boolean;
  sCellcheckboxcaption : string;
  sCellcheckboxinitval : boolean;
  sCellCombobox : boolean;
  sCellcomboboxstyle : TComboboxstyle;
  sCellButton : boolean;
  sCellButtoncaption : string;
  sCellSpinEdit : boolean;
  sCellspinvalue : longint;
end;
```

MakeNewSheet Method

The MakeNewSheet Method is new to version 2.5. Its syntax is:

Spread1.MakeNewSheet;

MakeNewSheet clears the entire spreadsheet of any cell data, special characteristics such as cell brushes, fonts, etc. and removes any objects such as bitmaps, checkboxes, buttons, etc.

Adding Components to the Grid

TSpread allows you to add other components to the grid, however they may be added only at run time. The types of components that can be added are:

Buttons
Check Boxes
Combo Boxes
Spin Edits

See Also:

Removing Components from the Grid
Accessing or Changing component properties
Naming Conventions

Adding Buttons to the Grid

You can add any number of buttons to the grid by calling the MakeButton method. The syntax for this method is:

Spread1.MakeButton(icol,irow,caption);

icol and irow are long integers and indicate what cell to place the button in. Caption is a string used to indicate the button caption.

See Also:

Controlling component events
Accessing or Changing component properties

Controlling Component Events

Once you've added a button, combobox, checkbox or spinedit to the grid you need a way to control what's happening. For example, if the user clicks a button you've added to the grid, you need to know it and you need to know which button was clicked. You can do this by adding your code to the event handlers to examine the state of check boxes, selected text of combo boxes, the value of a spinedit or monitor which buttons have clicked by the user.

For more information:

See events

Events

The following events are only fired in conjunction with components that have been added to the grid by one of the method calls.

OnButtonClick
OnComboChange
OnCheckClick
OnSpinchange
OnCellEnter
OnCellExit
OnRowEnter
OnRowExit
OnColEnter
OnColExit
OnCellTypeViol
OnCellEntryError

See Also:

Adding Buttons to the Grid
Adding Checkboxes to the Grid
Adding Comboboxes to the Grid
Adding SpinEdits to the Grid

OnButtonClick Event

The OnButtonClick event is fired when a button which has been added to the grid has been clicked by the user. The parameters passed to the event handler are col, row, name and caption. You can use any of these to identify which button triggered the event. Col and row indicate the cell location of the button, name represents the control name and caption is the button caption.

See Also:

Component naming conventions

Naming Conventions

When you add a component (buttons, check boxes or combo boxes) to the grid the component is named according to the col and row location. For example, a button added to the cell at column 10 and row 5 would be named 'Buttoncol10row5'. The other allowable components are similarly names 'Comboboxcol10row5' and 'checkboxcol10row5'.

See Also:

Adding Components to the Grid

OnComboChange Event

The OnCombochange event is fired when a selection has been made from a combo box which has been added to the grid. The parameters passed to the event handler are col, row, name and item. You can use any of these to identify which combo triggered the event. Col and row indicate the cell location of the combo, name represents the control name and item is the selected text from the combo box.

See Also:

Component Naming Conventions

OnCheckClick Event

The OnCheckClick event is fired when the user clicks a check box which has been added to the grid. The paramaters passed to the event handler are col, row, name and value. Col and row indicate which cell, name is the name of the check box which was clicked and value is a boolean true or false indicating the state of the check box's checked property.

See Also:

Component Naming Conventions

OnSpinChange Event

The OnSpinchange event is fired when the user changes the value in a spin edit which has been added to the grid. The parameters passed to the event handler are col, row, name and value. You can use any of these to identify which spinedit triggered the event. Col and row indicate the cell location of the spinedit, name represents the control name and value is the value of the spinedit box (value property). This event is fired whenever the user clicks on the up or down arrow or types in the edit box of the spin edit control.

See Also:

Component Naming Conventions

OnCellEnter Event

The OnCellEnter event is fired when the user selects a new cell from the grid. The handler declaration looks like this:

```
procedure TForm1.Spread1CellEnter(oldcol, oldrow, newcol,
   newrow: Longint; var confirm: Boolean);
```

oldcol and oldrow are the col and row reference for the current cell.
newcol and newrow are the col and row reference for the new cell being selected.
confirm is a boolean which if set to false in your handler code, will prevent the new cell from being selected. Note that confirm is initially true on handler entry.

The purpose of this handler is to allow cell data validation in a fashion similar to the onselectcell handler.

Note:

The OnCellExit event is fired before the OnCellEnter event.

See Also:

OnCellExit Event

OnCellExit Event

The OnCellExit event is fired when the user selects a new cell from the grid. The handler declaration looks like this:

```
procedure TForm1.Spread1CellExit(oldcol, oldrow, newcol,
   newrow: Longint; var confirm: Boolean);
```

oldcol and oldrow are the col and row reference for the current cell.
newcol and newrow are the col and row reference for the new cell being selected.
confirm is a boolean which if set to false in your handler code, will prevent the new cell from being selected. Note that confirm is initially true on handler entry.

The purpose of this handler is to allow cell data validation in a fashion similar to the onselectcell handler.

Note:

The OnCellExit event is fired before the OnCellEnter event.

See Also:

OnCellEnter Event

OnRowEnter Event

The OnRowEnter event is fired whenever the newly selected cell is in a different row in the grid that the previous cell.

The declaration looks like this:

```
procedure TForm1.Spread1RowEnter(oldrow, newrow: Longint;
   var confirm: Boolean);
```

oldrow is the row number of the previous cell.
newrow is the row being selected.
confirm is true upon handler entry, if you don't want to allow a row change, you can set this variable to false and oldrow will remain the current row after the handler exits.

See Also:

OnRowExit Event

OnRowExit Event

The OnRowExit event is fired whenever the newly selected cell is in a different row in the grid that the previous cell.

The declaration looks like this:

```
procedure TForm1.Spread1RowExit(oldrow, newrow: Longint;
   var confirm: Boolean);
```

oldrow is the row number of the previous cell.
newrow is the row being selected.
confirm is true upon handler entry, if you don't want to allow a row change, you can set this variable to false and oldrow will remain the current row after the handler exits.

See Also:

OnRowEnter Event

OnColEnter Event

The OnColEnter event is fired whenever the newly selected cell is in a different column in the grid that the previous cell.

The declaration looks like this:

```
procedure TForm1.Spread1ColEnter(oldcol, newcol: Longint;
   var confirm: Boolean);
```

oldcol is the row number of the previous cell.
newcol is the row being selected.
confirm is true upon handler entry, if you don't want to allow a column change, you can set this variable to false and oldcol will remain the current column after the handler exits.

See Also:

OnColExit Event

OnColExit Event

The OnColExit event is fired whenever the newly selected cell is in a different column in the grid that the previous cell.

The declaration looks like this:

```
procedure TForm1.Spread1ColExit(oldcol, newcol: Longint;
   var confirm: Boolean);
```

oldcol is the row number of the previous cell.
newcol is the row being selected.
confirm is true upon handler entry, if you don't want to allow a column change, you can set this variable to false and oldcol will remain the current column after the handler exits.

See Also:

OnColEnter Event

OnCellEntryError Event

The OnCellEntryError Event is fired whenever an entry in a cell is invalid. This could be because it is an invalid formula or an invalid type for the cell type assigned to that particular cell. When an invalid data type is entered the OnCellTypeViol event will be fired first. Refer to that discussion.

The declaration of this event handler looks like this:

```
procedure TForm1.Spread1CellEntryError(icol, irow: Longint;
   Celltype: Tctype; Cellcontents: String; var NewCellcontents: OpenString);
```

icol and irow are the column and row of the offending cell.
celltype is the cell type of the offending cell.
cellcontents is the current contents of the cell.
newcellcontents will be an empty string when the handler is invoked.

If you want to display something in the offending cell such as 'ERR' set newcellcontents to that in the handler and the component will place that string in the cell when the handler returns.

Adding Check Boxes to the Grid

You can add any number of checkboxes to the grid by calling the MakeCheckBox method. The syntax for this method is:

Spread1.MakeCheckBox(icol,irow,caption,initval);

icol and irow are long integers and indicate what cell to place the check box in. Caption is a string used to indicate the checkbox caption. Initval is a boolean used to set the initial state to checked (true) or unchecked (false).

See Also:

Controlling Component events
Accessing or Changing component properties

Accessing or Changing component properties

Once you have added components to the grid, you'll undoubtedly have a need to access a property or change the value of a property. The following example(s) apply to all components added to the grid.

To access the component, you'll need to declare a variable of the type of the component. Then call the Get... method of the TSpread component. This method returns the type of the component. You may then access or change the properties as needed.

Example 1 - Accessing combo box properties...

Assume you have added a combo box to a grid in column 1, row 1. The following code will allow you access to the properties.

```
var
     T : TCombobox;     {A variable of the type we need}
     icol, irow : longint;
begin
      icol := 1;
      irow := 1;
      T := Spread1.GetCellCombobox(icol,irow); {get the combo object}
      If T <> nil then   {always test for nil}
      begin
          T.Font.style := T.Font.style + [fsbold];   {change font to bold}
          T.items.add('one');        {add items to combo}
          T.items.add('two');
          T.items.add('three');
      end;
end;
```

Another Combo example... To get the itemindex of the combo item selected by the user:

```
var
     i : integer;
     T : Tcombobox;
begin
     T := Spread1.GetCellCombobox(Spread1.col,Spread1.row);
     If T <> nil then
        i := T.itemindex;
end;
```

Example 2 - Accessing checkbox properties...

{at this point you have already called Tspread's makecheckbox method to create the checkbox.}

```
var
     icol,irow : longint;
     T : TCheckbox;
begin
     icol := 0;
     irow := 0;
     T := Spread1.GetCellCheckbox(icol,irow);
     If T <> nil then
     begin
         If T.checked then
            T.caption := 'True'
         else
```

```
            T.caption := 'False';

      end;
end;
```

{a simple example that gets the value of the checked property and changes the value of the caption property}

Example 3 - Accessing button properties...

Let's combine a couple of things in this example... A check box and button have been added to the grid. If the checkbox is false we want to disable the button.

```
var
   btn : TButton;
   ck : Tcheckbox;
begin
      ck := Spread1.GetCellcheckbox(0,0);{assume checkbox is in 0,0}
      If ck <> nil then {always check for nil}
      begin
          btn := Spread1.GetCellbutton(1,0);
          If btn <> nil then
             btn.enabled :=     ck.checked;
      end;
end;
```

Example 4 - Accessing Spinedit properties...

This is all boilerplate stuff, one more example...

```
var
    v : integer;
    S : TSpinedit   {don't forget to add spin to your uses clause}
begin
    S := Spread1.GetCellSpinEdit(1,1);
    If S <> nil then
       v := S.value;     {gets the value of the spin}
end;
```

Adding Combo Boxes to the Grid

You can add any number of combo boxes to the grid by calling the MakeComboBox method. The syntax for this method is:

Spread1.MakeComboBox(icol,irow,style);

icol and irow are long integers and indicate what cell to place the check box in. The style argument is of type TComboBoxStyle. The valid values for this argument are csDropdownlist, csSimple, csDropdown, csOwnerdrawfixed and csOwnerdrawvariable.

Note:

The value of the Spread1.cells[col,row] becomes the item selected by the user in the combo, so you can either query that of the text property of the combo itselft. See the following topics below for some examples.

See Also:

Controlling Component events
Accessing or Changing component properties
Populating Combo Boxes

Populating Combo Boxes

Once you have added a combo box to the grid, you'll undoubtedly want to populate it with a list of strings. To do this, call the AddtoCombo method.

The syntax for this method is:

Spread1.Addtocombo(col,row,item);

col and row are long integers indicating the cell where the combo is located. Item is a string argument indicating the string to add.

Note:

This method has been left in for backward compatibility purposes. It is no longer needed in version 2.0 or later.

See Also:

Removing items from a combo box
Accessing or Changing component properties

Removing Items from a Combo Box

To remove an item from a combo box that has been added to the grid, call the removefromcombo method. The syntax is:

Spread1.removefromcombo(col,row,index);

col and row are long integers indicating the cell location of the combo box.
index is an integer indicating the index of the item to be removed.

Note:

This method has been left in for backward compatibility and is no longer needed.

See Also:

Populating Combo Boxes
Accessing or Changing component properties

Adding SpinEdits to the Grid

You can add any number of spin edits to the grid by calling the MakeSpinEdit method. The syntax for this method is:

Spread1.MakeSpinEdit(icol,irow,value);

icol and irow are long integers and indicate what cell to place the check box in. The value argument is of type longint and represents the initial value to assign to the spin edit's value property.

Note:

When the value of the spinedit is changed by the user, so is the cells[col,row] property of the underlying cell.

See Also:

Controlling Component Events

Accessing or Changing component properties

Removing Components from the Grid

To remove components that have been added to the grid, call one of the following methods:

RemoveButton
Removecheckbox
Removecombobox
RemoveSpinEdit

RemoveButton Method

Use the removebutton method to remove a button from the grid. The syntax is:

Spread1.Removebutton(col,row);

col and row are long integers indicating the cell location of the button.

RemoveCheckbox Method

Use the removecheckbox method to remove a checkbox from the grid. The syntax is:

Spread1.Removecheckbox(col,row);

col and row are long integers indicating the cell location of the checkbox.

RemoveCombobox Method

Use the removecombobox method to remove a combobox from the grid. The syntax is:

Spread1.Removecombobox(col,row);

col and row are long integers indicating the cell location of the combobox.

RemoveSpinEdit Method

Use the removespinedit method to remove a spin edit from the grid. The syntax is:

Spread1.RemoveSpinEdit(col,row);

col and row are long integers indicating the cell location of the spin edit.

Version 2.0

There are so many changes in version 2.0 that it would take a week to write about all of them. The most notable are the introduction of the concept of cell types. Version 2.0 corrects the problem of adding and deleting rows and columns from the grid. Cell frames are introduced in version 2.0 as well as a myriad of new methods.

Note:

Beginning with version 2.0 a formula must start with '=' unless it is a function such as @sum.

See Also:

Cell types
Using cell types
@Date function
@Time function
Accessing / Changing component properties
AddRow Method
DeleteRow Method
AddCol Method
DeleteCol Method
SetCellFrame Method
LockCell Method
LockAllCells Method
LockCol Method
LockRow Method
UnLockCell Method
UnLockAllCells Method
UnLockCol Method
UnLockRow Method
IsCellLocked Method
OnCellTypeViol Event
OnCellEntryError Event

Registration

The unregistered version of TSpread includes all the functionality of the registered version except it will only allow you to put one bitmap, button, check box, combo box or spinedit into a grid. The number is not limited in the registered version.

There are two ways to register TSpread. If you are a compuserve member you can register on line. GO SWREG # 9122 to register and I will file mail you a registered version as soon as compuserve notifies me of your registration.

If you are not a compuserve member you can register by mailing me a check for $ 30. Mail checks to:

John C. Taylor
3475 Holcomb Br Rd
Suite 202
Norcross, GA 30092

My phone is : (770)-449-6284

If you are mailing me a check I'll send you the registered copy as soon as you have notified me that you're sending me the check. Please indicate the check number in your correspondence. I prefer to be contacted email rather than voice mail. My email address is:

compuserve 76350,3301
Internet: 76350.3301@compuserve.com

I can distribute the registered version as a binhex encoded file to you via internet or I'll mail you a diskette (IF ABSOLUTELY NECESSARY). I would rather distribute via the Internet 'cause it's just a lot easier. I automatically send updates and bug fixes to registered users.

See Also:

Registering by Credit Card
How to Get the Source Code

Visa and MasterCard Registrations

I have contracted with NorthStar Solutions to process registrations via your valid Visa or MasterCard. NorthStar Solutions can be contacted via:

Voice: 1-800-699-6395 (10:00 am - 10:00 pm EST US callers only)
1-803-699-6395 (10:00 am - 10:00 pm EST outside US)

FAX: 1-803-699-5465 (Available 24 hours. International orders encouraged)

When contacting NorthStar, tell them you are registering John Taylor's TSpread component for Delphi. Have your credit card number and expiration date handy.

Note:

The component and the source code are separate items. If you want both be sure to say you want the Tspread component AND the source code. The charge for both items will be $ 100.00. (Source code $ 70 + component $ 30).

Important Note:

NorthStar Solutions cannot and will not provide technical support or other technical information about TSpread. Please contact them only for credit card registrations.

For technical support contact the author.

Technical Support
Source Code

Product Support

You can obtain support for TSpread through Compuserve or the Internet. If you have a problem, bug report or other comments and suggestions, please email me at:

compuserve : 76350,3301
internet: 76350.3301@compuserve.com

It is difficult to do support over the phone (voice) so please email me your problems, etc. I am very interested in suggestions and constructive criticism so please let me hear from you. If you find a bug please let me know so I can fix it !

Source Code

The source code for TSpread is available for an additional fee of $ 70. I think this is dirt cheap and once you have the source you can modify it any way you want. If you are a compuserve member you can register the source code separately, GO SWREG # 10358. You can register by credit card or you can send me a check. If paying by check, I will email you the source code when I receive the check. See the registration topic for details on where to send the check or how to register by credit card. If you register and receive the source code you agree to the following terms and conditions:

You Agree that:

You can customize and modify the TSpread source code in any way you wish. You may distribute any application that uses TSpread or any customized version of TSpread royalty free. You cannot create a component that uses TSpread as an ancestor and distribute that component. You cannot customize TSpread and distribute that component.

In a nutshell, you can't use my efforts to create a component to compete with TSpread.

Note:

TSpread uses a separate component for it's calculation engine. While that component is distributed free with TSpread, it's source code is not a part of the TSpread source code package and is not available.

Registration
Registering by credit card

Revision History

See Version 2.0 changes

Version 2.01

Corrects a bug which caused a GPF or other error message when setting grid options from the object inspector.

Version 2.5

Corrects several reported bugs. The calculation engine is much improved and yields a substantial increase (up to 20 fold in many cases) in calculation speed.

New Methods in Version 2.5

LoadFromFile Method
SaveToFile Method
MakeNewSheet Method

formula
A mathematical expression which begins with '=' and may contain variable names.

TAlignment

Predefined Delphi type, can be TaLeftJustify, TaRightJustify or TaCenter.

TcFrame

TcFrame type - valid values are cfTop, cfleft, cfright, cfbottom, cfoutline or cfnone

Tctype

Cell type - Valid values are ctnone, ctdate, cttime, ctformula, cttext or ctobject.

TFloatFormat

Predefined Delphi type, can be ffFixed, ffNumber, ffCurrency, ffGeneral or ffExponent.