

## Properties

[A](#)

[B](#)

[C](#)

[D](#)

[E](#)

[F](#)

[G](#)

[H](#)

[I](#)

[J](#)

[K](#)

[L](#)

[M](#)

[N](#)

[O](#)

[P](#)

[Q](#)

[R](#)

[S](#)

[T](#)

[U](#)

[V](#)

[W](#)

[X](#)

[Y](#)

[Z](#)

[Using the Property Topics](#)

[Understanding Syntax Conventions](#)

[Setting Enumerated Properties](#)

[Setting Properties for OCX Controls](#)

[Loading Pictures in C++](#)

## A

[About](#)

[Action](#)

[ActiveCol](#)

[ActiveRow](#)

[AllowCellOverflow](#)

[AllowDragDrop](#)

[AllowMultiBlocks](#)  
[AllowUserFormulas](#)  
[ArrowsExitEditMode](#)  
[AutoCalc](#)  
[AutoClipboard](#)  
[AutoSize](#)

## **B**

[BackColor](#)  
[BackColorStyle](#)  
[BlockMode](#)  
[BorderStyle](#)  
[ButtonDrawMode](#)

## **C**

[CellBorderColor](#)  
[CellBorderStyle](#)  
[CellBorderType](#)  
[CellType](#)  
[ChangeMade](#)  
[Clip](#)  
[ClipValue](#)  
[Col](#)  
[Col2](#)  
[ColHeaderDisplay](#)  
[ColHidden](#)  
[ColPageBreak](#)  
[ColsFrozen](#)  
[ColWidth](#)  
[CursorIcon](#)  
[CursorStyle](#)  
[CursorType](#)

## **D**

[DataChanged](#)  
[DataColCnt](#)  
[DataField](#)  
[DataFillEvent](#)  
[DataRowCnt](#)  
[DataSource](#)  
[DAutoCellTypes](#)  
[DAutoFill](#)  
[DAutoHeadings](#)  
[DAutoSave](#)  
[DAutoSizeCols](#)  
[DestCol](#)

[DestRow](#)  
[DInformActiveRowChange](#)  
[DisplayColHeaders](#)  
[DisplayRowHeaders](#)  
[DragIcon](#)  
[DragMode](#)

## **E**

[EditEnterAction](#)  
[EditMode](#)  
[EditModePermanent](#)  
[EditModeReplace](#)  
[Enabled](#)

## **F**

[FileNum](#)  
[FloatDefCurrencyChar](#)  
[FloatDefDecimalChar](#)  
[FloatDefSepChar](#)  
[Font](#)  
[FontBold](#)  
[FontItalic](#)  
[FontName](#)  
[FontSize](#)  
[FontStrikethru](#)  
[FontUnderline](#)  
[ForeColor](#)  
[Formula](#)  
[FormulaSync](#)

## **G**

[GrayAreaBackColor](#)  
[GridColor](#)  
[GridShowHoriz](#)  
[GridShowVert](#)  
[GridSolid](#)

## **H**

[hDCPrinter](#)  
[Height](#)  
[HelpContextID](#)  
[hWnd](#)

## **I**

[Index](#)  
[IsBlockSelected](#)

## **L**

[Left](#)

[LeftCol](#)

[Lock](#)

[LockBackColor](#)

[LockForeColor](#)

## **M**

[MaxCols](#)

[MaxRows](#)

[MaxTextCellHeight](#)

[MaxTextCellWidth](#)

[MaxTextColWidth](#)

[MaxTextRowHeight](#)

[MoveActiveOnFocus](#)

[MultiSelCount](#)

[MultiSelIndex](#)

## **N**

[Name](#)

[NoBeep](#)

[NoBorder](#)

## **O**

[OperationMode](#)

## **P**

[Parent](#)

[Position](#)

[PrintAbortMsg](#)

[PrintBorder](#)

[PrintColHeaders](#)

[PrintColor](#)

[PrintFooter](#)

[PrintGrid](#)

[PrintHeader](#)

[PrintJobName](#)

[PrintMarginBottom](#)

[PrintMarginLeft](#)

[PrintMarginRight](#)

[PrintMarginTop](#)

[PrintOrientation](#)

[PrintPageEnd](#)

[PrintPageStart](#)

[PrintRowHeaders](#)

[PrintShadows](#)

[PrintType](#)

[PrintUseDataMax](#)

[ProcessTab](#)

[Protect](#)

## R

[ReDraw](#)

[RestrictCols](#)

[RestrictRows](#)

[RetainSelBlock](#)

[Row](#)

[Row2](#)

[RowHeaderDisplay](#)

[RowHeight](#)

[RowHidden](#)

[RowPageBreak](#)

[RowsFrozen](#)

## S

[ScrollBarExtMode](#)

[ScrollBarMaxAlign](#)

[ScrollBars](#)

[ScrollBarShowMax](#)

[SelBlockCol](#)

[SelBlockCol2](#)

[SelBlockRow](#)

[SelBlockRow2](#)

[SelectBlockOptions](#)

[SelLength](#)

[SelModelIndex](#)

[SelModeSelCount](#)

[SelModeSelected](#)

[SelStart](#)

[SelText](#)

[ShadowColor](#)

[ShadowDark](#)

[ShadowText](#)

[SortBy](#)

[SortKey](#)

[SortKeyOrder](#)

[StartingColNumber](#)

[StartingRowNumber](#)

## T

[TabIndex](#)

[TabStop](#)

[Tag](#)

[Text](#)

[Top](#)

[TopRow](#)

[TypeButtonAlign](#)

[TypeButtonBorderColor](#)

[TypeButtonColor](#)

[TypeButtonDarkColor](#)

[TypeButtonLightColor](#)

[TypeButtonPicture](#)

[TypeButtonPictureDown](#)

[TypeButtonShadowSize](#)

[TypeButtonText](#)

[TypeButtonTextColor](#)

[TypeButtonType](#)

[TypeCheckCenter](#)

[TypeCheckPicture](#)

[TypeCheckText](#)

[TypeCheckTextAlign](#)

[TypeComboBoxCount](#)

[TypeComboBoxCurSel](#)

[TypeComboBoxEditable](#)

[TypeComboBoxIndex](#)

[TypeComboBoxList](#)

[TypeComboBoxString](#)

[TypeDateCentury](#)

[TypeDateFormat](#)

[TypeDateMax](#)

[TypeDateMin](#)

[TypeDateSeparator](#)

[TypeEditCharCase](#)

[TypeEditCharSet](#)

[TypeEditLen](#)

[TypeEditMultiLine](#)

[TypeEditPassword](#)

[TypeFloatCurrencyChar](#)

[TypeFloatDecimalChar](#)

[TypeFloatDecimalPlaces](#)

[TypeFloatMax](#)

[TypeFloatMin](#)

[TypeFloatMoney](#)

[TypeFloatSeparator](#)

[TypeFloatSepChar](#)

[TypeHAlign](#)

[TypeIntegerMax](#)

[TypeIntegerMin](#)

[TypeIntegerSpinInc](#)  
[TypeIntegerSpinWrap](#)  
[TypeOwnerDrawStyle](#)  
[TypePicDefaultText](#)  
[TypePicMask](#)  
[TypePictCenter](#)  
[TypePictMaintainScale](#)  
[TypePictPicture](#)  
[TypePictStretch](#)  
[TypeSpin](#)  
[TypeTextAlignVert](#)  
[TypeTextPrefix](#)  
[TypeTextShadow](#)  
[TypeTextShadowIn](#)  
[TypeTextWordWrap](#)  
[TypeTime24Hour](#)  
[TypeTimeMax](#)  
[TypeTimeMin](#)  
[TypeTimeSeconds](#)  
[TypeTimeSeparator](#)

## U

[UnitType](#)  
[UserResize](#)  
[UserResizeCol](#)  
[UserResizeRow](#)

## V

[Value](#)  
[VirtualCurRowCount](#)  
[VirtualCurTop](#)  
[VirtualMaxRows](#)  
[VirtualMode](#)  
[VirtualOverlap](#)  
[VirtualRows](#)  
[VirtualScrollBuffer](#)  
[Visible](#)  
[VisibleCols](#)  
[VisibleRows](#)  
[VScrollSpecial](#)  
[VScrollSpecialType](#)

## W

[Width](#)

No properties begin with this letter.



## Using the Property Topics

The following table describes each section in the property topics.

Topic Section	Description																	
Summary Box	<p>Illustrates</p> <ul style="list-style-type: none"> <li>Whether the property can be set using the Spread Designer</li> <li>Whether the property is available at run time, design time, or both</li> <li>Whether the property is read/write, read-only, or write-only</li> </ul>																	
	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td></td> <td style="text-align: center;">Read</td> <td style="text-align: center;">Write</td> <td style="text-align: center;">Run Time</td> <td style="text-align: center;">Design Time</td> </tr> <tr> <td style="text-align: center;">Spread Designer</td> <td style="text-align: center;">SD</td> <td style="text-align: center;">RD</td> <td style="text-align: center;">WR</td> <td style="text-align: center;">RT</td> <td style="text-align: center;">DT</td> </tr> <tr> <td></td> <td style="text-align: center;">✓</td> <td style="text-align: center;">✓</td> <td style="text-align: center;">✓</td> <td style="text-align: center;">✓</td> <td style="text-align: center;">✓</td> </tr> </table>		Read	Write	Run Time	Design Time	Spread Designer	SD	RD	WR	RT	DT		✓	✓	✓	✓	✓
	Read	Write	Run Time	Design Time														
Spread Designer	SD	RD	WR	RT	DT													
	✓	✓	✓	✓	✓													
Description	Provides a brief summary of the property																	
Syntax	Shows how to set or read the value of the property in code (See <a href="#">Understanding Syntax Conventions</a> for information about typographic usage in syntax statements.)																	
OCX Use	Lists the OCX property page and item corresponding to the property, if applicable																	
Remarks	Provides detailed and related information, including enumerated property settings and constants																	
Data Type	Indicates the Visual Basic data type of the property's values																	
Example	<p>Provides C++ and Visual Basic example code for the property</p> <p><b>Note</b> Properties that are not set in the examples use their default values. To determine the default value for a property, refer to its description.</p>																	
See Also	Provides references to related information																	

## Understanding Syntax Conventions

The "Syntax" section uses the typographic conventions listed in the following table.

Example	Description
<i>value, color</i>	Italicized items are placeholders for information you supply.
<b>short, BOOL</b>	Bold, italicized arguments indicate data types, pointers, and user-defined types.
[ = value& ]	Items inside square brackets are optional.

## Setting Enumerated Properties

Enumerated properties are those with a finite number of settings that can be specified with whole numbers (integers) or constants. For example, the [ScrollBars](#) property can be set to None (0 or SS\_SCROLLBAR\_NONE), Horizontal (1 or SS\_SCROLLBAR\_H\_ONLY), Vertical (2 or SS\_SCROLLBAR\_V\_ONLY), or Both (3 or SS\_SCROLLBAR\_BOTH). When setting enumerated properties in code, you can use either of the following methods:

- Set the property to an integer value, for example,
 

```
vaSpread1.ScrollBars = 2
```
- Set the property to a constant, for example,
 

```
vaSpread1.ScrollBars = SS_SCROLLBAR_V_ONLY
```

Using constants makes your code easier to read and requires fewer comments. In addition, using constants ensures compatibility with future versions of the Spread control.

Constants are defined for all Spread enumerated properties in constants files. The Setup program copies constants files into the \INCLUDE subdirectory. To use constants in your code, you must include the constants files in your applications.

## Loading Pictures in C++

If you are using the C++ programming language and loading pictures in the Spread VBX control, you must use additional code before setting picture properties.

Use the following code as an example of how to convert an hBitmap into an hPic (the bitmap is created in AppStudio).

```
// Convert hBitmap to hPic for the VBX using C++
PIC picture;
HPIC hPic;
HBITMAP hBitmap;

hBitmap=LoadBitmap(AfxGetInstanceHandle( ), "bitmap");
picture.picData.bmp.hbitmap=hBitmap;
picture.picType=PICTURE_BITMAP;
hPic=AfxSetPict(NULL, &picture);
vaSpread1->SetTypePictPicture(hPic);
```

## Setting Properties for OCX Controls

The Spread OCX control presents properties available for the control on *property pages*. Each page represents an aspect of the control, such as edit mode or headers features. Properties that are not available on property pages can be set either through the property browser, through the Spread Designer, or in code.

The property descriptions present information about the properties represented by the property page items. For each property represented on a property page, the section "OCX Use" describes which item on which page represents the property.

For more information about using the Spread OCX control, see [Using OCX Controls](#).

SD	RD	WR	RT	DT
	✓		✓	✓

## About Property

### Description

Returns version information at run time. At design time, double-clicking this property displays a dialog box with version information.

### Syntax

C++ (MFC VBX)    **CString** CSpreadSheet::GetAbout( );

C++ (OWL VBX)    **string** CSpreadSheet::GetAbout( );

Visual Basic    [form.]Spread.About

### Remarks

The About dialog box displays the Spread version number, control name, and version date. At run time, you can retrieve the version number as a string and can convert the string to a floating-point number. The version number format is "V.RR", where "V" is the one-digit version number from 0 through 9, and "RR" is the two-digit update or revision number from 00 through 99.

### Data Type

String

SD	RD	WR	RT	DT
✓		✓	✓	

## Action Property

[See Also](#)    [Example](#)

### Description

Performs a specified action based on the setting assigned. This property is available at run time only.

### Syntax

C++                    **void** CSpreadSheet::SetAction(**short** value);

Visual Basic        [form.]Spread.Action[ = setting%]

### Remarks

The following settings are available:

Setting	Constant Description
0 - Activate Cell	SS_ACTION_ACTIVE_CELL Sets the active cell (Use the <a href="#">Col</a> and <a href="#">Row</a> properties to specify the active cell.)
1 - Go To	SS_ACTION_GOTO_CELL Goes to the specified cell without changing the active cell (Use the <a href="#">Col</a> , <a href="#">Row</a> , and <a href="#">Position</a> properties with this setting.)
2 - Select Block	SS_ACTION_SELECT_BLOCK Selects a block of cells (Use the <a href="#">Col</a> , <a href="#">Row</a> , <a href="#">Col2</a> , and <a href="#">Row2</a> properties with this setting.)
3 - Clear	SS_ACTION_CLEAR Clears a cell, a block of cells, a column, a row, or the entire spreadsheet (Use the <a href="#">Col</a> , <a href="#">Row</a> , <a href="#">Col2</a> , and <a href="#">Row2</a> properties with this setting.)
4 - Delete Col	SS_ACTION_DELETE_COL Deletes a column (Use the <a href="#">Col</a> property to specify the column to delete.) <b>Note</b> Deleting columns does not decrease the number of columns set by the <a href="#">MaxCols</a> property. When you use the Action property to delete a column, a blank column is automatically added at the end of the spreadsheet.
5 - Delete Row	SS_ACTION_DELETE_ROW Deletes a row (Use the <a href="#">Row</a> property to specify the row to delete.) <b>Note</b> Deleting rows does not decrease the number of rows set by the <a href="#">MaxRows</a> property. When you use the Action property to delete a row, a blank row is automatically added at the end of the spreadsheet.
6 - Insert Col	SS_ACTION_INSERT_COL Inserts a column before the specified column (Use the <a href="#">Col</a> property to specify where to insert the new column.) <b>Caution</b> Inserting columns does not increase the number of columns set by the <a href="#">MaxCols</a> property. To avoid losing data, increase the value of the MaxCols property before inserting columns.
7 - Insert Row	SS_ACTION_INSERT_ROW Inserts a row before the specified row (Use the <a href="#">Row</a> property to specify where to insert the new row.) <b>Caution</b> Inserting rows does not increase the number of rows set by the <a href="#">MaxRows</a> property. To avoid losing data, increase the value of the MaxRows property before

	inserting rows.
8 - Load	<p>SS_ACTION_LOAD_SPREAD_SHEET  Loads spreadsheet data  (Use the <a href="#">FileNum</a> property to specify the file to load.)  <b>Note</b> This setting is available for VBX controls in Visual Basic and Microsoft Visual C++ only. OCX and Borland C++ users must use the <a href="#">LoadFromFile</a> or <a href="#">LoadTabFile</a> method or function.</p>
9 - Save All	<p>SS_ACTION_SAVE_ALL  Saves the current spreadsheet  (Use the <a href="#">FileNum</a> property to specify the file to save.)  <b>Note</b> This setting is available for VBX controls in Visual Basic and Microsoft Visual C++ only. OCX and Borland C++ users must use the <a href="#">SaveToFile</a> method or function.</p>
10 - Save Values	<p>SS_ACTION_SAVE_VALUES  Saves only the text of the current spreadsheet  (Use the <a href="#">FileNum</a> property to specify the file to save.)  <b>Note</b> This setting is available for VBX controls in Visual Basic and Microsoft Visual C++ only. OCX and Borland C++ users must use the <a href="#">SaveTabFile</a> method or function.</p>
11 - Recalculate	<p>SS_ACTION_RECALC  Performs manual recalculation on every formula</p>
12 - Clear Text	<p>SS_ACTION_CLEAR_TEXT  Removes only data from a cell, a column, a row, a block of cells, or the entire spreadsheet  (Use the <a href="#">Col</a>, <a href="#">Row</a>, <a href="#">Col2</a>, and <a href="#">Row2</a> properties with this setting.)</p>
13 - Print	<p>SS_ACTION_PRINT  Prints the spreadsheet  (Use the associated printing properties, such as <a href="#">PrintBorder</a>, to specify printing options.)</p>
14 - Deselect Block	<p>SS_ACTION_DESELECT_BLOCK  Deselects all selected blocks when the spreadsheet is in normal operation mode</p>
15 - Data Save	<p>SS_ACTION_DSAVE  Saves all records in a bound Spread control that have been modified  (This setting is valid only if the <a href="#">DAutoSave</a> property is set to False.)</p>
16 - Set Border	<p>SS_ACTION_SET_CELL_BORDER  Sets a border around a cell or block of cells  (Use the <a href="#">CellBorderColor</a>, <a href="#">CellBorderStyle</a>, and <a href="#">CellBorderType</a> properties to specify border options.)</p>
17 - Add MultiSel Blocks	<p>SS_ACTION_ADD_MULTISELBLOCK  Selects a block of cells if the spreadsheet is in normal operation mode and the <a href="#">AllowMultiBlocks</a> property is set to True  (Use the <a href="#">Col</a>, <a href="#">Row</a>, <a href="#">Col2</a>, <a href="#">Row2</a>, and <a href="#">BlockMode</a> properties to specify the block.)</p>
18 - Get MultiSel Blocks	<p>SS_ACTION_GET_MULTI_SELECTION  Records the selected blocks of cells if the spreadsheet is in normal operation mode and the <a href="#">AllowMultiBlocks</a> property is set to True  (Access the recorded information using the <a href="#">MultiSelCount</a> and <a href="#">MultiSelIndex</a> properties. The blocks are indexed from largest to smallest—not in the order in which you select them.)</p>
19 - Copy Range	<p>SS_ACTION_COPY_RANGE  Copies a block of cells to another location within the spreadsheet  (Use the <a href="#">Col</a>, <a href="#">Row</a>, <a href="#">Col2</a>, <a href="#">Row2</a>, <a href="#">DestCol</a>, and <a href="#">DestRow</a> properties with this setting.)</p>
20 - Move Range	<p>SS_ACTION_MOVE_RANGE  Moves a block of cells to another location within the spreadsheet  (Use the <a href="#">Col</a>, <a href="#">Row</a>, <a href="#">Col2</a>, <a href="#">Row2</a>, <a href="#">DestCol</a>, and <a href="#">DestRow</a> properties with this setting.)</p>

21 - Swap Range	<p>SS_ACTION_SWAP_RANGE          Swaps a block of cells with another block of cells within the spreadsheet          (Use the <a href="#">Col</a>, <a href="#">Row</a>, <a href="#">Col2</a>, <a href="#">Row2</a>, <a href="#">DestCol</a>, and <a href="#">DestRow</a> properties with this setting.)</p>
22 - Clipboard Copy	<p>SS_ACTION_CLIPBOARD_COPY          Copies the data from the selected block of cells to the Clipboard</p>
23 - Clipboard Cut	<p>SS_ACTION_CLIPBOARD_CUT          Cuts the data from the selected block of cells to the Clipboard</p>
24 - Clipboard Paste	<p>SS_ACTION_CLIPBOARD_PASTE          Pastes the data from the Clipboard to the selected block of cells</p>
25 - Sort	<p>SS_ACTION_SORT          Sorts the specified cells within the spreadsheet          (Use the <a href="#">Col</a>, <a href="#">Row</a>, <a href="#">Col2</a>, and <a href="#">Row2</a> properties to specify the cells and the <a href="#">SortBy</a>, <a href="#">SortKey</a>, and <a href="#">SortKeyOrder</a> properties to specify sorting options.)</p>
26 - Combo Clear	<p>SS_ACTION_COMBO_CLEAR          Clears the contents (all items) in a combo box cell</p>
27 - Combo Remove String	<p>SS_ACTION_COMBO_REMOVE          Removes an item specified by the <a href="#">TypeComboBoxIndex</a> property from a combo box cell</p>
28 - Reset	<p>SS_ACTION_RESET          Clears and resets all attributes of the spreadsheet back to the default settings</p>
29 - Select Mode Clear	<p>SS_ACTION_SEL_MODE_CLEAR          Deselects all the selections of a spreadsheet in multiple-selection or extended-selection operation mode</p>
30 - VRefresh Buffer	<p>SS_ACTION_VMODE_REFRESH          Forces the spreadsheet to discard current data and re-request the data currently in the buffer          (Use this setting with the <a href="#">VirtualMode</a> property.)</p>
32 - Smart Print	<p>SS_ACTION_SMARTPRINT          Prints the spreadsheet using Smart Printing provided by the spreadsheet          The following rules apply:</p> <ul style="list-style-type: none"> <li>▪ If the spreadsheet is wider than a portrait page, the spreadsheet prints in landscape mode.</li> <li>▪ If the information does not fit in landscape mode, but will fit in landscape mode if the spreadsheet is reduced up to 60% of its original size, the spreadsheet is scaled to fit within the page.</li> <li>▪ If the information cannot be scaled to fit, the spreadsheet tries to reduce column widths to accommodate the widest string within each column.</li> <li>▪ If all attempts to make the spreadsheet print within a page fail, printing resumes normally in the current printer orientation with no reductions.</li> </ul>

**Data Type**

Integer (Enumerated)

[Example - Action setting 0 \(Activate Cell\)](#)  
[Example - Action setting 1 \(Go To\)](#)  
[Example - Action setting 2 \(Select Block\)](#)  
[Example - Action setting 3 \(Clear\)](#)  
[Example - Action setting 4 \(Delete Col\)](#)  
[Example - Action setting 5 \(Delete Row\)](#)  
[Example - Action setting 6 \(Insert Col\)](#)  
[Example - Action setting 7 \(Insert Row\)](#)  
[Example - Action setting 8 \(Load\)](#)  
[Example - Action setting 9 \(Save All\)](#)  
[Example - Action setting 10 \(Save Values\)](#)  
[Example - Action setting 11 \(Recalculate\)](#)  
[Example - Action setting 12 \(Clear Text\)](#)  
[Example - Action setting 13 \(Print\)](#)  
[Example - Action setting 16 \(Set Border\)](#)  
[Example - Action setting 17 \(Add MultiSel Blocks\)](#)  
[Example - Action setting 18 \(Get MultiSel Blocks\)](#)  
[Example - Action setting 19 \(Copy Range\)](#)  
[Example - Action setting 20 \(Move Range\)](#)  
[Example - Action setting 21 \(Swap Range\)](#)  
[Example - Action setting 25 \(Sort\)](#)  
[Example - Action setting 26 \(Combo Clear\)](#)  
[Example - Action setting 27 \(Combo Remove String\)](#)



[Print](#)

[Copy](#)

[Close](#)

#### **Example—Action 0**

The following example sets the Position property to upper left and goes to the cell at column 5, row 5.

C++

```
// Set the active cell and scroll the spreadsheet to the
// active cell
vaSpread1->SetRow(5);
vaSpread1->SetCol(5);
vaSpread1->SetPosition(SS_POSITION_UPPER_LEFT);
vaSpread1->SetAction(SS_ACTION_ACTIVE_CELL);
vaSpread1->SetAction(SS_ACTION_GOTO_CELL);
```

Visual Basic

```
' Set the active cell and scroll the spreadsheet to the
' active cell
vaSpread1.Row = 5
vaSpread1.Col = 5
vaSpread1.Position = SS_POSITION_UPPER_LEFT
vaSpread1.Action = SS_ACTION_ACTIVE_CELL
vaSpread1.Action = SS_ACTION_GOTO_CELL
```

[Print](#)

[Copy](#)

[Close](#)

#### **Example—Action 1**

The following example sets the [Position](#) property to upper left and goes to the cell at column 5, row 5.

C++

```
// Set the active cell and scroll the spreadsheet to the
// active cell
vaSpread1->SetRow(5);
vaSpread1->SetCol(5);
vaSpread1->SetPosition(SS_POSITION_UPPER_LEFT);
vaSpread1->SetAction(SS_ACTION_GOTO_CELL);
```

Visual Basic

```
' Set the active cell and scroll the spreadsheet to the
' active cell
vaSpread1.Row = 5
vaSpread1.Col = 5
vaSpread1.Position = SS_POSITION_UPPER_LEFT
vaSpread1.Action = SS_ACTION_GOTO_CELL
```

[Print](#)

[Copy](#)

[Close](#)

**Example—Action 2**

The following example selects a block of cells.

C++

```
// Select a block of cells
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
vaSpread1->SetRow2(4);
vaSpread1->SetCol2(4);
vaSpread1->SetAction(SS_ACTION_SELECT_BLOCK);
```

Visual Basic

```
' Select a block of cells
vaSpread1.Row = 2
vaSpread1.Col = 2
vaSpread1.Row2 = 4
vaSpread1.Col2 = 4
vaSpread1.Action = SS_ACTION_SELECT_BLOCK
```

[Print](#)

[Copy](#)

[Close](#)

### Example—Action 3

The following example clears a block of cells, deleting both the data and the format.

C++

```
// Select a block of cells
vaSpread1->SetRow(4);
vaSpread1->SetCol(3);
vaSpread1->SetRow2(8);
vaSpread1->SetCol2(4);
vaSpread1->SetBlockMode(TRUE);
// Clear the data and format of the cells
vaSpread1->SetAction(SS_ACTION_CLEAR);
// Turn block mode off
vaSpread1->SetBlockMode(FALSE);
```

Visual Basic

```
' Select a block of cells
vaSpread1.Row = 4
vaSpread1.Col = 3
vaSpread1.Row2 = 8
vaSpread1.Col2 = 4
vaSpread1.BlockMode = True
' Clear the data and format of the cells
vaSpread1.Action = SS_ACTION_CLEAR
' Turn block mode off
vaSpread1.BlockMode = False
```

[Print](#)

[Copy](#)

[Close](#)

**Example—Action 4**

The following example deletes column 3 from the spreadsheet.

C++

```
// Select a column
vaSpread1->SetCol(3);
// Delete the selected column
vaSpread1->SetAction(SS_ACTION_DELETE_COL);
```

Visual Basic

```
' Select a column
vaSpread1.Col = 3
' Delete the selected column
vaSpread1.Action = SS_ACTION_DELETE_COL
```

[Print](#)

[Copy](#)

[Close](#)

#### **Example—Action 5**

The following example deletes row 4 from the spreadsheet.

C++

```
// Select a row
vaSpread1->SetRow(4);
// Delete the selected row
vaSpread1->SetAction(SS_ACTION_DELETE_ROW);
```

Visual Basic

```
' Select a row
vaSpread1.Row = 4
' Delete the selected row
vaSpread1.Action = SS_ACTION_DELETE_ROW
```

[Print](#)

[Copy](#)

[Close](#)

### **Example—Action 6**

The following example inserts a column into the spreadsheet before column 2.

C++

```
// Select a column  
vaSpread1->SetCol(2);  
// Insert a column before the selected column  
vaSpread1->SetAction(SS_ACTION_INSERT_COL);
```

Visual Basic

```
' Select a column  
vaSpread1.Col = 2  
' Insert a column before the selected column  
vaSpread1.Action = SS_ACTION_INSERT_COL
```

[Print](#)

[Copy](#)

[Close](#)

**Example—Action 7**

The following example inserts a row into the spreadsheet before row 2.

C++

```
// Select a row
vaSpread1->SetRow(2);
// Insert a row before the selected row
vaSpread1->SetAction(SS_ACTION_INSERT_ROW);
```

Visual Basic

```
' Select a row
vaSpread1.Row = 2
' Insert a row before the selected row
vaSpread1.Action = SS_ACTION_INSERT_ROW
```



[Print](#)

[Copy](#)

[Close](#)

**Example ▪ Action 8**

The following example loads an existing spreadsheet file.

Visual Basic (OCX)

```
Dim rc As Integer
rc = vaSpread1.LoadFromFile("presaved.ss2")
```

Visual Basic (VBX)

```
SS_FileName = CMD.FileName
Open CMD.FileName For Binary As #1
vaSpread1.FileNum = 1
vaSpread1.Action = SS_ACTION_LOAD_SPREAD_SHEET
Close #1
vaSpread1.ChangeMade = False
```

[Print](#)

[Copy](#)

[Close](#)

**Example ▪ Action 9**

The following example saves the entire spreadsheet to a file.

Visual Basic (OCX)

```
Dim rc As Integer  
rc = vaSpread1.SaveToFile("saveall.ss2", False)
```

Visual Basic (VBX)

```
Open SS_FileName For Binary As #1  
vaSpread1.FileNum = 1  
vaSpread1.Action = SS_ACTION_SAVE_ALL  
Close #1  
vaSpread1.ChangeMade = False
```

[Print](#)

[Copy](#)

[Close](#)

**Example ▪ Action 10**

The following example saves only the values in the spreadsheet to a file.

Visual Basic (OCX)

```
Dim rc As Integer
rc = vaSpread1.SaveToFile("dataonly.ss2", True)
```

Visual Basic (VBX)

```
Open SS_FileName For Binary As #1
vaSpread1.FileNum = 1
vaSpread1.Action = SS_ACTION_SAVE_VALUES
Close #1
vaSpread1.ChangeMade = False
```

[Print](#)

[Copy](#)

[Close](#)

**Example ▪ Action 11**

The following example forces all formulas in the spreadsheet to be recalculated.

C++

```
vaSpread1->SetAction(SS_ACTION_RECALC);
```

Visual Basic

```
vaSpread1.Action = SS_ACTION_RECALC
```

[Print](#)

[Copy](#)

[Close](#)

**Example ▪ Action 12**

The following example clears a block of cells. Only the data is deleted and not the format.

C++

```
// Select a block of cells
vaSpread1->SetRow(4);
vaSpread1->SetCol(3);
vaSpread1->SetRow2(8);
vaSpread1->SetCol2(4);
vaSpread1->SetBlockMode(TRUE);
// Clear the data from the cells
vaSpread1->SetAction(SS_ACTION_CLEAR_TEXT);
// Turn block mode off
vaSpread1->SetBlockMode(FALSE);
```

Visual Basic

```
' Select a block of cells
vaSpread1.Row = 4
vaSpread1.Col = 3
vaSpread1.Row2 = 8
vaSpread1.Col2 = 4
vaSpread1.BlockMode = True
' Clear the data from the cells
vaSpread1.Action = SS_ACTION_CLEAR_TEXT
' Turn block mode off
vaSpread1.BlockMode = False
```

[Print](#)

[Copy](#)

[Close](#)

### Example ▪ Action 13

The following example illustrates how to print a spreadsheet.

C++

```
// Set printing options for spreadsheet
vaSpreadl->SetPrintAbortMsg("Printing - Click Cancel to quit");
vaSpreadl->SetPrintJobName("XYZ Software");
vaSpreadl->SetPrintHeader("/cPrint Header/rPage #/p/n2nd Line");
vaSpreadl->SetPrintFooter("/cPrint Footer/rPage #/p/n2nd Line");
vaSpreadl->SetPrintBorder(TRUE);
vaSpreadl->SetPrintColHeaders(FALSE);
vaSpreadl->SetPrintColor(TRUE);
vaSpreadl->SetPrintGrid(FALSE);
vaSpreadl->SetPrintMarginTop(1440);
vaSpreadl->SetPrintMarginBottom(1440);
vaSpreadl->SetPrintMarginLeft(720);
vaSpreadl->SetPrintMarginRight(720);
vaSpreadl->SetPrintType(SS_PRINT_ALL);
vaSpreadl->SetPrintRowHeaders(FALSE);
vaSpreadl->SetPrintShadows(FALSE);
vaSpreadl->SetPrintUseDataMax(TRUE);
// Perform the printing action
vaSpreadl->SetAction(SS_ACTION_PRINT);
```

Visual Basic

```
' Set printing options for spreadsheet
vaSpreadl.PrintAbortMsg = "Printing - Click Cancel to quit"
vaSpreadl.PrintJobName = "XYZ Software"
vaSpreadl.PrintHeader = "/cPrint Header/rPage #/p/n2nd Line"
vaSpreadl.PrintFooter = "/cPrint Footer/rPage #/p/n2nd Line"
vaSpreadl.PrintBorder = True
vaSpreadl.PrintColHeaders = False
vaSpreadl.PrintColor = True
vaSpreadl.PrintGrid = False
vaSpreadl.PrintMarginTop = 1440
vaSpreadl.PrintMarginBottom = 1440
vaSpreadl.PrintMarginLeft = 720
vaSpreadl.PrintMarginRight = 720
vaSpreadl.PrintType = SS_PRINT_ALL
vaSpreadl.PrintRowHeaders = False
vaSpreadl.PrintShadows = False
vaSpreadl.PrintUseDataMax = True
' Perform the printing action
vaSpreadl.Action = SS_ACTION_PRINT
```

[Print](#)

[Copy](#)

[Close](#)

**Example ▪ Action 16**

The following example sets a cell's border to outline.

C++

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Turn cell outline borders on
vaSpread1->SetCellBorderType(SS_BORDER_TYPE_OUTLINE);
vaSpread1->SetCellBorderStyle(SS_BORDER_STYLE_SOLID);
// black, RGB(0,0,0)
vaSpread1->SetCellBorderColor(0x00000000);
vaSpread1->SetAction(SS_ACTION_SET_CELL_BORDER);
```

Visual Basic

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Turn cell outline borders on
vaSpread1.CellBorderType = SS_BORDER_TYPE_OUTLINE
vaSpread1.CellBorderStyle = SS_BORDER_STYLE_SOLID
' black, RGB(0,0,0)
vaSpread1.CellBorderColor = &H00000000&
vaSpread1.Action = SS_ACTION_SET_CELL_BORDER
```

[Print](#)

[Copy](#)

[Close](#)

**Example ▪ Action 17**

The following example adds a cell block selection to a spreadsheet that allows multiple block selections.

C++

```
vaSpread1->SetRow(2);  
vaSpread1->SetCol(2);  
vaSpread1->SetRow2(4);  
vaSpread1->SetCol2(4);  
vaSpread1->SetBlockMode(TRUE);  
vaSpread1->SetAction(SS_ACTION_ADD_MULTISELBLOCK);  
vaSpread1->SetBlockMode(FALSE);
```

Visual Basic

```
vaSpread1.Row = 2  
vaSpread1.Col = 2  
vaSpread1.Row2 = 4  
vaSpread1.Col2 = 4  
vaSpread1.BlockMode = True  
vaSpread1.Action = SS_ACTION_ADD_MULTISELBLOCK  
vaSpread1.BlockMode = False
```



[Print](#)

[Copy](#)

[Close](#)

#### Example ▪ Action 18

The following example determines whether there is a multiple cell block selection and, if so, sets each cell block selection's background color to blue; otherwise, the single cell background color is turned red.

C++

```
// Determine if a multiple cell block selection exists
vaSpread1->SetAction(SS_ACTION_GET_MULTI_SELECTION);
// Set whether a block of cells is selected
if(vaSpread1->IsBlockSelected( ) || vaSpread1->MultiSelCount( ))
{
    int i;
    // Set the background color for each cell block
    vaSpread1->SetBlockMode(TRUE);
    for(i = 0; i < vaSpread1->MultiSelCount( ); i++)
    {
        vaSpread1->SetMultiSelIndex(i);
        // blue, RGB(0,0,255)
        vaSpread1->SetBackColor(0x00FF0000);
    }
    vaSpread1->SetBlockMode(FALSE);
}
// If a single cell was selected
else

// Set the background color for the cell
vaSpread1->SetCol(vaSpread1.GetActiveCol( ));
vaSpread1->SetRow(vaSpread1.GetActiveRow( ));
// red, RGB(255,0,0)
vaSpread1->SetBackColor(0x000000FF);
endif
// Turn the redrawing for the spreadsheet back on
vaSpread1->SetReDraw(TRUE);
// Deselect block so blue appears (not inverse)
vaSpread1->SetAction(SS_ACTION_DESELECT_BLOCK);
```

Visual Basic

```
Dim i As Integer
' Determine if a multiple selection exists
vaSpread1.Action = SS_ACTION_GET_MULTI_SELECTION

' If a block of cells was selected
If vaSpread1.IsBlockSelected Or vaSpread1.MultiSelCount Then
    ' Set the background color for each cell block
    vaSpread1.BlockMode = True
    For i = 0 To vaSpread1.MultiSelCount - 1
        vaSpread1.MultiSelIndex = i
        ' blue, RGB(0,0,255)
        vaSpread1.BackColor = &H00FF0000&
    Next i
    vaSpread1.BlockMode = False
' If a single cell was selected
Else
    ' Set the background color for the cell
    vaSpread1.Col = vaSpread1.ActiveCol
    vaSpread1.Row = vaSpread1.ActiveRow
    ' red, RGB(255,0,0)
    vaSpread1.BackColor = &H000000FF&
EndIf
' Turn the redrawing for the spreadsheet back on
vaSpread1.ReDraw = True
' Deselect block so blue appears (not inverse)
vaSpread1.Action = SS_ACTION_DESELECT_BLOCK
```



[Print](#)

[Copy](#)

[Close](#)

**Example ▪ Action 19**

The following example copies a block of cells from one location to another.

C++

```
// Select a block of cells
vaSpread1->SetRow(3);
vaSpread1->SetCol(2);
vaSpread1->SetRow2(7);
vaSpread1->SetCol2(2);
// Copy the selected block of cells
vaSpread1->SetDestCol(5);
vaSpread1->SetDestRow(3);
vaSpread1->SetAction(SS_ACTION_COPY_RANGE);
```

Visual Basic

```
' Select a block of cells
vaSpread1.Row = 3
vaSpread1.Col = 2
vaSpread1.Row2 = 7
vaSpread1.Col2 = 2
' Copy the selected block of cells
vaSpread1.DestCol = 5
vaSpread1.DestRow = 3
vaSpread1.Action = SS_ACTION_COPY_RANGE
```

[Print](#)

[Copy](#)

[Close](#)

**Example ▪ Action 20**

The following example moves a block of cells from one location to another.

C++

```
// Select a block of cells
vaSpread1->SetRow(3);
vaSpread1->SetCol(2);
vaSpread1->SetRow2(7);
vaSpread1->SetCol2(2);
// Move the selected block of cells
vaSpread1->SetDestCol(5);
vaSpread1->SetDestRow(3);
vaSpread1->SetAction(SS_ACTION_MOVE_RANGE);
```

Visual Basic

```
' Select a block of cells
vaSpread1.Row = 3
vaSpread1.Col = 2
vaSpread1.Row2 = 7
vaSpread1.Col2 = 2
' Move the selected block of cells
vaSpread1.DestCol = 5
vaSpread1.DestRow = 3
vaSpread1.Action = SS_ACTION_MOVE_RANGE
```

[Print](#)

[Copy](#)

[Close](#)

**Example ▪ Action 21**

The following example swaps the contents of two blocks of cells.

C++

```
// Select a block of cells
vaSpread1->SetRow(3);
vaSpread1->SetCol(2);
vaSpread1->SetRow2(7);
vaSpread1->SetCol2(2);
// Swap with another block of cells
vaSpread1->SetDestCol(4);
vaSpread1->SetDestRow(3);
vaSpread1->SetAction(SS_ACTION_SWAP_RANGE);
```

Visual Basic

```
' Select a block of cells
vaSpread1.Row = 3
vaSpread1.Col = 2
vaSpread1.Row2 = 7
vaSpread1.Col2 = 2
' Swap with another block of cells
vaSpread1.DestCol = 4
vaSpread1.DestRow = 3
vaSpread1.Action = SS_ACTION_SWAP_RANGE
```

[Print](#)

[Copy](#)

[Close](#)

#### **Example ▪ Action 25**

The following example sorts data in a selected block of cells.

**C++**

```
// Select a block of cells
vaSpread1->SetRow(3);
vaSpread1->SetCol(2);
vaSpread1->SetRow2(11);
vaSpread1->SetCol2(3);
// Set sort definition for key 1
vaSpread1->SetSortBy(SS_SORT_BY_ROW);
vaSpread1->SetSortKey(1,2);
vaSpread1->SetSortKeyOrder(1,SS_SORT_ORDER_ASCENDING);
// Set sort definition for key 2
vaSpread1->SetSortKey(2,3);
vaSpread1->SetSortKeyOrder(2,SS_SORT_ORDER_DESCENDING);
vaSpread1->SetAction(SS_ACTION_SORT);
```

**Visual Basic**

```
' Select a block of cells
vaSpread1.Row = 3
vaSpread1.Col = 2
vaSpread1.Row2 = 11
vaSpread1.Col2 = 3
' Set sort definition for key 1
vaSpread1.SortBy = SS_SORT_BY_ROW
vaSpread1.SortKey(1) = 2
vaSpread1.SortKeyOrder(1) = SS_SORT_ORDER_ASCENDING
' Set sort definition for key 2
vaSpread1.SortKey(2) = 3
vaSpread1.SortKeyOrder(2) = SS_SORT_ORDER_DESCENDING
vaSpread1.Action = SS_ACTION_SORT
```

[Print](#)

[Copy](#)

[Close](#)

**Example ▪ Action 26**

The following example removes all items from a combo box cell.

C++

```
vaSpread1->SetCol(2);  
vaSpread1->SetRow(2);  
vaSpread1->SetAction(SS_ACTION_COMBO_CLEAR);
```

Visual Basic

```
vaSpread1.Col = 2  
vaSpread1.Row = 2  
vaSpread1.Action = SS_ACTION_COMBO_CLEAR
```

[Print](#)

[Copy](#)

[Close](#)

**Example ▪ Action 27**

The following example removes the second item in the combo box list.

C++

```
vaSpread1->SetCol(2);  
vaSpread1->SetRow(2);  
vaSpread1->SetTypeComboBoxIndex(1);  
vaSpread1->SetAction(SS_ACTION_COMBO_REMOVE);
```

Visual Basic

```
vaSpread1.Col = 2  
vaSpread1.Row = 2  
vaSpread1.TypeComboBoxIndex = 1  
vaSpread1.Action = SS_ACTION_COMBO_REMOVE
```



## See Also

[AllowMultiBlocks](#), [BlockMode](#), [CellBorderColor](#), [CellBorderStyle](#), [CellBorderType](#), [Col](#), [Col2](#), [DAutoSave](#), [DestCol](#), [DestRow](#), [FileNum](#), [MaxCols](#), [MaxRows](#), [MultiSelCount](#), [MultiSelIndex](#), [Position](#), [PrintBorder](#), [Row](#), [Row2](#), [SortBy](#), [SortKey](#), [SortKeyOrder](#), [TypeComboBoxIndex](#), [VirtualMode](#) properties

[PrintAbort](#) event

[LoadFromFile](#), [LoadTabFile](#), [SaveTabFile](#), [SaveToFile](#) functions

SD	RD	WR	RT	DT
	✓		✓	

## ActiveCol Property

[See Also](#)

[Example](#)

### Description

Returns the column number of the active cell. This property is available at run time only.

### Syntax

C++ (OCX)      **long** CSpreadSheet::GetActiveCol( );  
C++ (VBX)      **short** CSpreadSheet::GetActiveCol( );  
Visual Basic    *[form.]Spread.ActiveCol*

### Remarks

The default value for the ActiveCol property is 1.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example places the text "Hello" in the active cell.

C++

```
// Select the active cell
vaSpread1->SetRow(vaSpread1->GetActiveRow( ));
vaSpread1->SetCol(vaSpread1->GetActiveCol( ));
// Enter data into the cell
vaSpread1->SetText("Hello");
```

Visual Basic

```
' Select the active cell
vaSpread1.Row = vaSpread1.ActiveRow
vaSpread1.Col = vaSpread1.ActiveCol
' Enter data into the cell
vaSpread1.Text = "Hello"
```

**See Also**

[Action](#) (0 - Activate Cell), [ActiveRow](#) properties

SD	RD	WR	RT	DT
	✓		✓	

## ActiveRow Property

[See Also](#)

[Example](#)

### Description

Returns the row number of the active cell. This property is available at run time only.

### Syntax

C++ **long** CSpreadSheet::GetActiveRow( );

Visual Basic [form.]Spread.ActiveRow

### Remarks

The default value for the ActiveRow property is 1.

### Data Type

Long Integer

**See Also**

[Action](#) (0 - Activate Cell), [ActiveCol](#) properties

SD	RD	WR	RT	DT
✓	✓	✓	✓	✓

## AllowCellOverflow Property

### [Example](#)

### Description

Sets or returns whether text can overflow to adjacent empty cells.

### Syntax

C++ **BOOL** CSpreadSheet::GetAllowCellOverflow( );  
**void** CSpreadSheet::SetAllowCellOverflow(**BOOL** value);

Visual Basic [form.]Spread.AllowCellOverflow[ = boolean%]

### OCX Use

Cell Overflow check box under Allow on the [General](#) property page.

### Remarks

The default value for the AllowCellOverflow property is False.

When the AllowCellOverflow property is set to True,

- Left-aligned text in a cell overflows to the adjacent right cell.
- Right-aligned text in a cell overflows to the adjacent left cell.
- When text is centered in a cell, text overflows to both the left and right adjacent cells.

**Note** When the AllowCellOverflow property is set to True, the spreadsheet operates more slowly.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example lets text overflow into adjacent empty cells.

C++

```
// Allow text to overflow into the next empty cell  
vaSpread1->SetAllowCellOverflow(TRUE);
```

Visual Basic

```
' Allow text to overflow into the next empty cell  
vaSpread1.AllowCellOverflow = True
```



SD	RD	WR	RT	DT
✓	✓	✓	✓	✓

## AllowDragDrop Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the user can place a selected block of cells by dragging them to a new location in the same spreadsheet.

### Syntax

C++ **BOOL** CSpreadSheet::GetAllowDragDrop( );  
**void** CSpreadSheet::SetAllowDragDrop(**BOOL** value);

Visual Basic [form.]Spread.AllowDragDrop[ = boolean%]

### OCX Use

Drag and Drop check box under Allow on the [General](#) property page

### Remarks

The default value for the AllowDragDrop property is False.

When the AllowDragDrop property is set to True and the user releases the mouse button, the [DragDropBlock](#) event occurs.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example lets the user drag and drop a selected block of cells to a new location.

C++

```
// Allow drag-and-drop operations  
vaSpread1->SetAllowDragDrop(TRUE);
```

Visual Basic

```
' Allow drag-and-drop operations  
vaSpread1.AllowDragDrop = True
```

**See Also**

[DragDropBlock](#) event

SD	RD	WR	RT	DT
✓	✓	✓	✓	✓

## AllowMultiBlocks Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the user can select multiple, noncontiguous blocks of cells.

### Syntax

C++                    **BOOL** CSpreadSheet::GetAllowMultiBlocks( );  
                         **void** CSpreadSheet::SetAllowMultiBlocks(**BOOL** *value*);

Visual Basic        [form.]Spread.AllowMultiBlocks[ = *boolean%*]

### OCX Use

Multiple Block Selections check box under Allow on the [General](#) property page

### Remarks

The default value for the AllowMultiBlocks property is False.

When the AllowMultiBlocks property is set to True, the user can select multiple, noncontiguous blocks of cells.

You must use a mouse to select multiple blocks of cells. Select a cell or a block of cells by pressing the left mouse button on a cell and dragging. To select a second block of cells, press the Ctrl key during subsequent cell block selections.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example lets the user select multiple, noncontiguous cells in the same spreadsheet.

C++

```
// Allow multiple block selections  
vaSpread1->SetAllowMultiBlocks(TRUE);
```

Visual Basic

```
' Allow multiple block selections  
vaSpread1.AllowMultiBlocks = True
```

**See Also**

[Action](#) (14 - Deselect Block, 17 - Add MultiSel Blocks, 18 - Get MultiSel Blocks), [MultiSelCount](#), [MultiSelIndex](#) properties

SD	RD	WR	RT	DT
✓	✓	✓	✓	✓

## AllowUserFormulas Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the user can type formulas into float or integer cells.

### Syntax

C++ **BOOL** CSpreadSheet::GetAllowUserFormulas( );  
**void** CSpreadSheet::SetAllowUserFormulas(**BOOL** value);

Visual Basic [form.]Spread.AllowUserFormulas[ = boolean%]

### OCX Use

Users to Enter Formulas check box under Allow on the [General](#) property page

### Remarks

The default value for the AllowUserFormulas property is False.

When the AllowUserFormulas property is set to True, the user can press the Equal Sign (=) key to type, display, or make modifications to formulas in float or integer cells.

This property has no effect on the application's use of the [Formula](#) property.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example lets the user type formulas into float or integer cells.

C++

```
// Allow user-entered formulas  
vaSpread1->SetAllowUserFormulas(TRUE);
```

Visual Basic

```
' Allow user-entered formulas  
vaSpread1.AllowUserFormulas = True
```



**See Also**

[Formula](#) property

[UserFormulaEntered](#) event

SD	RD	WR	RT	DT
✓	✓	✓	✓	✓

## ArrowsExitEditMode Property

[Example](#)

### Description

Sets or returns the function of the arrow keys during edit mode.

### Syntax

C++ **BOOL** CSpreadSheet::GetArrowsExitEditMode( );  
**void** CSpreadSheet::SetArrowsExitEditMode(**BOOL** value);

Visual Basic [form.]Spread.ArrowsExitEditMode[ = boolean%]

### OCX Use

Arrow Keys Exit check box under Edit Mode on the [Edit Mode](#) property page

### Remarks

The default value for the ArrowsExitEditMode property is False.

When the ArrowsExitEditMode property is set to True, pressing the arrow keys causes the cell to exit edit mode and move the input focus in the appropriate direction. When the ArrowsExitEditMode property is set to False, the arrow keys are used for editing the cell, and the user must press the Ctrl key and an arrow key to exit edit mode.

For more information about edit mode, see [Using Edit Mode](#).

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example configures the spreadsheet to exit edit mode and move the focus when the user presses an arrow key.

C++

```
vaSpread1->SetArrowsExitEditMode (TRUE) ;
```

Visual Basic

```
vaSpread1.ArrowsExitEditMode = True
```

SD	RD	WR	RT	DT
✓	✓	✓	✓	✓

## AutoCalc Property

[See Also](#)      [Example](#)

### Description

Sets or returns whether the spreadsheet recalculates each formula when the contents of dependent cells change.

### Syntax

C++                    **BOOL** CSpreadSheet::GetAutoCalc( );  
                         **void** CSpreadSheet::SetAutoCalc(**BOOL** value);

Visual Basic        [form.]Spread.AutoCalc[ = boolean%]

### OCX Use

Automatic Calculations check box under Calculations on the [General](#) property page

### Remarks

The default value for the AutoCalc property is True.

When the AutoCalc property is set to True, the spreadsheet automatically recalculates each formula when the contents of cells referenced by the formula change. When the AutoCalc property is set to False, cells with formulas are not updated when data changes.

You can use the [Action](#) property to force a recalculation of all formulas.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example prevents cells with formulas from updating when the data changes.

C++

```
vaSpread1->SetAutoCalc (FALSE) ;
```

Visual Basic

```
vaSpread1.AutoCalc = False
```

**See Also**

[Action](#) (11 - Recalculate) property

SD	RD	WR	RT	DT
✓	✓	✓	✓	✓

## AutoClipboard Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the spreadsheet supports the Clipboard shortcut keys.

### Syntax

C++                    **BOOL** CSpreadSheet::GetAutoClipboard( );  
                         **void** CSpreadSheet::SetAutoClipboard(**BOOL** value);

Visual Basic        [form.]Spread.AutoClipboard[ = boolean%]

### OCX Use

Handle Clipboard Keys check box on the [General](#) property page

### Remarks

The default value for the AutoClipboard property is True.

When the AutoClipboard property is set to True, the spreadsheet supports the Clipboard shortcut keys. When the AutoClipboard property is set to False, the spreadsheet does not support the Clipboard shortcut keys.

The Clipboard shortcut keys include Ctrl+X to cut selections, Ctrl+C to copy selections, and Ctrl+V to paste selections.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example disables support of the Clipboard shortcut keys.

C++

```
vaSpread1->SetAutoClipboard (FALSE);
```

Visual Basic

```
vaSpread1.AutoClipboard = False
```



**See Also**

[Action](#) (22 - Clipboard Copy, 23 - Clipboard Cut, 24 - Clipboard Paste) property

SD	RD	WR	RT	DT
✓	✓	✓	✓	✓

## AutoSize Property

[See Also](#)    [Example](#)

### Description

Sets or returns whether the spreadsheet is automatically sized to display only complete columns and rows.

### Syntax

```
C++                    BOOL CSpreadSheet::GetAutoSize( );  
                      void CSpreadSheet::SetAutoSize(BOOL value);  
  
Visual Basic         [form.]Spread.AutoSize[ = boolean%]
```

### OCX Use

Auto Size check box on the [Op.Mode](#) property page

### Remarks

The default value for the AutoSize property is False.

When the AutoSize property is set to True, the right and bottom borders of the spreadsheet are aligned against the column and row boundaries to prevent partial display of columns or rows.

When the [VisibleCols](#) and [VisibleRows](#) properties are set to values other than 0, they designate the number of columns and rows that are fully displayed. When the VisibleCols and VisibleRows properties are set to 0, the settings of the [MaxCols](#) and [MaxRows](#) properties are used to designate the number of columns and rows that are fully displayed.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example creates a spreadsheet with 10 visible rows and 4 visible columns that sizes to display only complete rows and columns.

C++

```
// Set the number of visible rows
vaSpread1->SetVisibleRows(10);
// Set the number of visible columns
vaSpread1->SetVisibleCols(4);
// Allow autoresizing of spreadsheet to fill parent
vaSpread1->SetAutoSize(TRUE);
```

Visual Basic

```
' Set the number of visible rows
vaSpread1.VisibleRows = 10
' Set the number of visible columns
vaSpread1.VisibleCols = 4
' Allow autoresizing of spreadsheet to fill parent
vaSpread1.AutoSize = True
```

**See Also**

[MaxCols](#), [MaxRows](#), [VisibleCols](#), [VisibleRows](#) properties

SD	RD	WR	RT	DT
✓	✓	✓	✓	

## BackColor Property

[See Also](#)

[Example](#)

### Description

Sets or returns the background color of a cell, a column, a row, a block of cells, or the entire spreadsheet. This property is available at run time only.

### Syntax

C++ (OCX)      **OLE\_COLOR** CSpreadSheet::GetBackColor( );  
**void** CSpreadSheet::SetBackColor(**OLE\_COLOR** value);

C++ (VBX)      **COLORREF** CSpreadSheet::GetBackColor( );  
**void** CSpreadSheet::SetBackColor(**COLORREF** value);

Visual Basic      [form.]Spread.BackColor[ = color]

### OCX Use

Select BackColor from the Property Name drop-down list box on the [Colors](#) property page. The BackColor property is a stock property.

### Remarks

The default value for the BackColor property is &H00FFFFFF& (white).

Use the [Col](#) and [Row](#) properties to specify a cell to which to apply a color before setting the BackColor property. To apply a color to a block of cells, use the Col, [Col2](#), Row, [Row2](#), and [BlockMode](#) properties to specify the block of cells before setting the BackColor property.

If you use the BackColor property to retrieve the background color for a block of cells that do not all have the same background color, the background color of the active cell is returned.

### Data Type

Color

[Print](#)

[Copy](#)

[Close](#)

The following example defines the background color as yellow for the cell in column 2, row 2.

C++

```
// Select a cell
vaSpread1->SetCol(2);
vaSpread1->SetRow(2);
// Define the background color
// yellow, RGB(255,255,0)
vaSpread1->SetBackColor(0x0000FFFF);
```

Visual Basic

```
' Select a cell
vaSpread1.Col = 2
vaSpread1.Row = 2
' Define the background color
' yellow, RGB(255,255,0)
vaSpread1.BackColor = &H0000FFFF&
```

**See Also**

[BackColorStyle](#), [BlockMode](#), [Col](#), [Col2](#), [ForeColor](#), [Row](#), [Row2](#) properties

## BackColorStyle Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the background colors of cells overlap the grid lines.

### Syntax

C++                    **short** CSpreadSheet::GetBackColorStyle( );  
                         **void** CSpreadSheet::SetBackColorStyle(**short** value);

Visual Basic        [form.]Spread.BackColorStyle[ = setting%]

### OCX Use

Back Color Displays drop-down list box under Grid Lines on the [Display](#) property page

### Remarks

The following settings are available:

Setting	Description	Constant
0 - Over Grid	(Default) Hides the right and bottom sides of the cells' grid lines under the background color	SS_BACKCOLORSTYLE_OVERGRID
1 - Under Grid	Displays complete grid lines	SS_BACKCOLORSTYLE_UNDERGRID
2 - Over Horizontal Grid Only	Hides the horizontal grid lines under the background color	SS_BACKCOLORSTYLE_OVERHORIZGRIDONLY
3 - Over Vertical Grid Only	Hides the vertical grid lines under the background color	SS_BACKCOLORSTYLE_OVERVERTGRIDONLY

### Data Type

Integer (Enumerated)



[Print](#)

[Copy](#)

[Close](#)

The following example sets the background color to yellow and sets the background color style to overlap the grid lines.

C++

```
// Set the background color style
vaSpread1->SetBackColorStyle(SS_BACKCOLORSTYLE_OVERGRID);
// Define the background color
// yellow, RGB(255,255,0)
vaSpread1->SetBackColor(0x0000FFFF);
```

Visual Basic

```
' Set the background color style
vaSpread1.BackColorStyle = SS_BACKCOLORSTYLE_OVERGRID
' Define the background color
' yellow, RGB(255,255,0)
vaSpread1.BackColor = &H0000FFFF&
```

**See Also**

[BackColor](#) property

SD	RD	WR	RT	DT
✓	✓	✓	✓	

## BlockMode Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether you can modify a block of cells. This property is available at run time only.

### Syntax

```
C++          BOOL CSpreadSheet::GetBlockMode( );
             void CSpreadSheet::SetBlockMode(BOOL value);

Visual Basic [form.]Spread.BlockMode[ = boolean%]
```

### Remarks

The BlockMode property lets you perform an action on, provide data for, or assign settings to a selected block of cells.

The default value for the BlockMode property is False, which means you can only modify individual cells.

Use the [Col](#), [Row](#), [Col2](#), and [Row2](#) properties to designate a block of cells in the spreadsheet. When the BlockMode property is set to True, you can make modifications to the designated block of cells using the following properties:

- Certain settings of the [Action](#) property:
  - 3 - Clear                   SS\_ACTION\_CLEAR
  - 4 - Delete Col            SS\_ACTION\_DELETE\_COL
  - 5 - Delete Row            SS\_ACTION\_DELETE\_ROW
  - 6 - Insert Col             SS\_ACTION\_INSERT\_COL
  - 7 - Insert Row             SS\_ACTION\_INSERT\_ROW
  - 12 - Clear Text            SS\_ACTION\_CLEAR\_TEXT
  - 16 - Set Border            SS\_ACTION\_SET\_CELL\_BORDER
  - 17 - Add MultiSel Blocks  SS\_ACTION\_ADD\_MULTISELBLOCK
- [BackColor](#)
- [CellBorderColor](#)
- [CellBorderStyle](#)
- [CellBorderType](#)
- [CellType](#) and CellType-related properties
- [Font](#) and Font-related properties
- [ForeColor](#)
- [Formula](#)
- [Lock](#)
- [Text](#)
- [Value](#)

**Note** When modifications to a block of cells are complete, the BlockMode property should be set to False.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example places the text "Hello" in the designated block of cells and then turns block mode off.

C++

```
// Select a block of cells
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
vaSpread1->SetRow2(4);
vaSpread1->SetCol2(4);
vaSpread1->SetBlockMode(TRUE);
// Set the text for the cells
vaSpread1->SetText("Hello");
// Turn block mode off
vaSpread1->SetBlockMode(FALSE);
```

Visual Basic

```
' Select a block of cells
vaSpread1.Row = 2
vaSpread1.Col = 2
vaSpread1.Row2 = 4
vaSpread1.Col2 = 4
vaSpread1.BlockMode = True
' Set the text for the cells
vaSpread1.Text = "Hello"
' Turn block mode off
vaSpread1.BlockMode = False
```

**See Also**

[Action](#), [BackColor](#), [CellBorderColor](#), [CellBorderStyle](#), [CellBorderType](#), [CellType](#), [Col](#), [Col2](#), [Font](#), [FontBold](#), [FontItalic](#), [FontName](#), [FontSize](#), [FontStrikethru](#), [FontUnderline](#), [ForeColor](#), [Formula](#), [Lock](#), [Row](#), [Row2](#), [Text](#), [Value](#) properties

<b>SD</b>	<b>RD</b>	<b>WR</b>	<b>RT</b>	<b>DT</b>
	✓	✓	✓	✓

## BorderStyle Property

### Description

Sets or returns the border style for the Spread control. This property is read-only at run time.

### Syntax

C++                    **short** CSpreadSheet::GetBorderStyle( );  
**void** CSpreadSheet::SetBorderStyle(**short** value);

Visual Basic        [form.]Spread.BorderStyle[ = setting%]

### OCX Use

The BorderStyle property is a stock property.

### Remarks

The following settings are available:

Setting	Description	Constants
0 - None	Does not display a border	NONE (VB3) vbBSNone (VB4)
1 - Fixed Single	(Default) Displays a single-line border around the control	FIXED_SINGLE (VB3) vbFixedSingle (VB4)

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

Integer (Enumerated)

## ButtonDrawMode Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether to display buttons in button and combo box cells.

### Syntax

C++                    **short** CSpreadSheet::GetButtonDrawMode( );  
                         **void** CSpreadSheet::SetButtonDrawMode(**short** value);

Visual Basic        [form.]Spread.ButtonDrawMode[ = value%]

### OCX Use

Always, Current Cell, Current Column, and Current Row check boxes under Button Draw Mode on the [Display](#) property page

### Remarks

The following values are available. Note that you can combine values 1 (Current Cell), 2 (Current Column), and 4 (Current Row) using the Or operator to limit where buttons are displayed in the spreadsheet.

Value	Description	Constant
0 - Always	(Default) Always displays buttons	SS_BDM_ALWAYS
1 - Current Cell	Displays buttons only in the current cell	SS_BDM_CURRENT_CELL
2 - Current Column	Displays buttons only in the current column	SS_BDM_CURRENT_COLUMN
4 - Current Row	Displays buttons only in the current row	SS_BDM_CURRENT_ROW

### Data Type

Integer

[Print](#)

[Copy](#)

[Close](#)

The following example configures the spreadsheet to display buttons at all times.

C++

```
vaSpread1->SetButtonDrawMode(SS_BDM_ALWAYS);
```

Visual Basic

```
vaSpread1.ButtonDrawMode = SS_BDM_ALWAYS
```



**See Also**

[CellType](#) property

SD	RD	WR	RT	DT
✓	✓	✓	✓	

## CellBorderColor Property

[See Also](#)

[Example](#)

### Description

Sets or returns the border color of a cell, a column, a row, a block of cells, or the entire spreadsheet. This property is available at run time only.

### Syntax

C++ (OCX)      **unsigned long** CSpreadSheet::GetCellBorderColor( );  
**void** CSpreadSheet::SetCellBorderColor(**unsigned long** value);

C++ (VBX)      **COLORREF** CSpreadSheet::GetCellBorderColor( );  
**void** CSpreadSheet::SetCellBorderColor(**COLORREF** value);

Visual Basic      [form.]Spread.CellBorderColor[ = color]

### OCX Use

Select CellBorderColor from the Property Name drop-down list box on the [Colors](#) property page.

### Remarks

Use the [Col](#) and [Row](#) properties to specify a cell to which to apply a border color before setting the CellBorderColor property. To apply a border color to a block of cells, use the Col, [Col2](#), Row, [Row2](#), and [BlockMode](#) properties to specify the block of cells before setting the CellBorderColor property.

Use the CellBorderColor property with the [Action](#) property.

The default value for the CellBorderColor property is &H00000000& (black).

If you use the CellBorderColor property to retrieve the border color for a block of cells that do not all have the same border color, the border color of the active cell is returned.

### Data Type

VBX: Color  
OCX: Unsigned Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets the cell border color for a cell to red.

C++

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Turn cell outline borders on
vaSpread1->SetCellBorderStyle(SS_BORDER_TYPE_OUTLINE);
vaSpread1->SetCellBorderStyle(SS_BORDER_STYLE_SOLID);
// red, RGB(255,0,0)
vaSpread1->SetCellBorderColor(0x000000FF);
vaSpread1->SetAction(SS_ACTION_SET_CELL_BORDER);
```

Visual Basic

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Turn cell outline borders on
vaSpread1.CellBorderStyle = SS_BORDER_TYPE_OUTLINE
vaSpread1.CellBorderStyle = SS_BORDER_STYLE_SOLID
' red, RGB(255,0,0)
vaSpread1.CellBorderColor = &H000000FF&
vaSpread1.Action = SS_ACTION_SET_CELL_BORDER
```

**See Also**

[Action](#) (16 - Set Border), [BlockMode](#), [CellBorderStyle](#), [CellBorderType](#), [Col](#), [Col2](#), [Row](#), [Row2](#) properties

SD	RD	WR	RT	DT
✓	✓	✓	✓	

## CellBorderStyle Property

[See Also](#)

[Example](#)

### Description

Sets or returns the border style of a cell, a column, a row, a block of cells, or the entire spreadsheet. This property is available at run time only.

### Syntax

C++ **short** CSpreadSheet::GetCellBorderStyle( );  
**void** CSpreadSheet::SetCellBorderStyle(**short** value);

Visual Basic [form.]Spread.CellBorderStyle[ = setting%]

### Remarks

Use the [Col](#) and [Row](#) properties to specify a cell to which to apply a border style before setting the CellBorderStyle property. To apply a border style to a block of cells, use the Col, [Col2](#), Row, [Row2](#), and [BlockMode](#) properties to specify the block of cells before setting the CellBorderStyle property.

Use the CellBorderStyle property with the [Action](#) property.

The following settings are available:

Setting	Description	Constant
0 - Default	(Default) Displays default grid around cell	SS_BORDER_STYLE_DEFAULT
1 - Solid	Displays solid border around cell	SS_BORDER_STYLE_SOLID
2 - Dash	Displays dashed border around cell	SS_BORDER_STYLE_DASH
3 - Dot	Displays dotted border around cell	SS_BORDER_STYLE_DOT
4 - Dash Dot	Displays dash dot border around cell	SS_BORDER_STYLE_DASH_DOT
5 - Dash Dot Dot	Displays dash dot dot border around cell	SS_BORDER_STYLE_DASH_DOT_DOT
6 - Blank	Removes border from cell	SS_BORDER_STYLE_BLANK
11 - Fine Solid	Displays thin solid border around cell	SS_BORDER_STYLE_FINE_SOLID
12 - Fine Dash	Displays thin dashed border around cell	SS_BORDER_STYLE_FINE_DASH
13 - Fine Dot	Displays thin dotted border around cell	SS_BORDER_STYLE_FINE_DOT
14 - Fine Dash Dot	Displays thin dash dot border around cell	SS_BORDER_STYLE_FINE_DASH_DOT
15 - Fine Dash Dot Dot	Displays thin dash dot dot border around cell	SS_BORDER_STYLE_FINE_DASH_DOT_DOT

**Note** Only one cell border style can be applied to any given cell.

If you use the CellBorderStyle property to retrieve the border style for a block of cells that do not all have the same border style, the border style of the active cell is returned.

### Data Type

Integer (Enumerated)

**See Also**

[Action](#) (16 - Set Border), [BlockMode](#), [CellBorderColor](#), [CellBorderType](#), [Col](#), [Col2](#), [Row](#), [Row2](#) properties

SD	RD	WR	RT	DT
✓	✓	✓	✓	

## CellBorderStyle Property

[See Also](#)    [Example](#)

### Description

Sets or returns whether to display a border around a cell, a column, a row, a block of cells, or the entire spreadsheet. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetCellBorderStyle( );  
**void** CSpreadSheet::SetCellBorderStyle(**short** value);

Visual Basic        [form.]Spread.CellBorderStyle[ = value%]

### Remarks

Use the [Col](#) and [Row](#) properties to specify a cell to which to apply a border type before setting the CellBorderStyle property. To apply a border type to a block of cells, use the Col, [Col2](#), Row, [Row2](#), and [BlockMode](#) properties to specify the block of cells before setting the CellBorderStyle property.

Use the CellBorderStyle property with the [Action](#) property.

The following values are available. Note that you can combine values 1 (Left), 2 (Right), 4 (Top), and 8 (Bottom) using the Or operator to customize the cell border type.

Value	Description	Constant
0 - None	(Default) Does not display border	SS_BORDER_TYPE_NONE
1 - Left	Displays the border on the left side of the cell	SS_BORDER_TYPE_LEFT
2 - Right	Displays the border on the right side of the cell	SS_BORDER_TYPE_RIGHT
4 - Top	Displays the border on the top of the cell	SS_BORDER_TYPE_TOP
8 - Bottom	Displays the border on the bottom of the cell	SS_BORDER_TYPE_BOTTOM
16 - Outline	Displays the border around a block of cells	SS_BORDER_TYPE_OUTLINE

**Note** Only one cell border type can be applied to any given cell.

If you use the CellBorderStyle property to retrieve the border type for a block of cells that do not all have the same border type, the border type of the active cell is returned.

### Data Type

Integer

**See Also**

[Action](#) (16 - Set Border), [BlockMode](#), [CellBorderColor](#), [CellBorderStyle](#), [Col](#), [Col2](#), [Row](#), [Row2](#) properties



SD	RD	WR	RT	DT
✓	✓	✓	✓	

## CellType Property

[See Also](#)    [Example](#)

### Description

Sets or returns the type of data displayed in a cell, a column, a row, a block of cells, or the entire spreadsheet. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetCellType( );  
**void** CSpreadSheet::SetCellType(**short** value);

Visual Basic        [form.]Spread.CellType[ = setting%]

### Remarks

The following settings are available:

Setting	Description	Constant
0 - Date	Creates date cell	SS_CELL_TYPE_DATE
1 - Edit	(Default) Creates edit cell	SS_CELL_TYPE_EDIT
2 - Float	Creates float cell	SS_CELL_TYPE_FLOAT
3 - Integer	Creates integer cell	SS_CELL_TYPE_INTEGER
4 - PIC	Creates PIC cell	SS_CELL_TYPE_PIC
5 - Static Text	Creates static text cell	SS_CELL_TYPE_STATIC_TEXT
6 - Time	Creates time cell	SS_CELL_TYPE_TIME
7 - Button	Creates button cell	SS_CELL_TYPE_BUTTON
8 - Combo Box	Creates combo box cell	SS_CELL_TYPE_COMBOBOX
9 - Picture	Creates picture cell	SS_CELL_TYPE_PICTURE
10 - Check Box	Creates check box cell	SS_CELL_TYPE_CHECKBOX
11 - Owner-Drawn	Creates owner-drawn cell	SS_CELL_TYPE_OWNER_DRAWN

Use the [Col](#) and [Row](#) properties to specify a cell to which to apply a cell type before setting the CellType property. To apply a cell type to a block of cells, use the Col, [Col2](#), Row, [Row2](#), and [BlockMode](#) properties to specify the block of cells before setting the CellType property.

To specify a cell type for the entire spreadsheet, set the Col or Row properties to - 1. To specify a cell type for an entire column, set the Col property to a non-zero value and the Row property to

- 1. To specify a cell type for an entire row, set the Row property to a non-zero value and the Col property to
- 1. Once you specify the cell type, you can set the properties associated with that type.

If you use the CellType property to retrieve the cell type for a block of cells that are not all the same type, the cell type of the active cell is returned.

### Data Type

Integer (Enumerated)

[Example - CellType setting 0 \(Date\)](#)

[Example - CellType setting 1 \(Edit\)](#)

[Example - CellType setting 2 \(Float\)](#)

[Example - CellType setting 3 \(Integer\)](#)

[Example - CellType setting 4 \(PIC\)](#)

[Example - CellType setting 5 \(Static Text\)](#)

[Example - CellType setting 6 \(Time\)](#)

[Example - CellType setting 7 \(Button\)](#)

[Example - CellType setting 8 \(Combo Box\)](#)

[Example - CellType setting 9 \(Picture\)](#)

[Example - CellType setting 10 \(Check Box\)](#)

[Example - CellType setting 11 \(Owner Drawn\)](#)

[Print](#)

[Copy](#)

[Close](#)

### Example—CellType 0

The following example sets a cell to be a date cell.

C++

```
// Select a cell
vaSpread1->SetCol(2);
vaSpread1->SetRow(2);
// Define cell as type DATE
vaSpread1->SetCellType(SS_CELL_TYPE_DATE);
vaSpread1->SetTypeDateCentury(FALSE);
vaSpread1->SetTypeHAlign(SS_CELL_H_ALIGN_LEFT);
vaSpread1->SetTypeSpin(FALSE);
vaSpread1->SetTypeDateFormat(SS_CELL_DATE_FORMAT_MMDDYY);
vaSpread1->SetTypeDateMin("01011990");
vaSpread1->SetTypeDateMax("01011995");
```

Visual Basic

```
' Select a cell
vaSpread1.Col = 2
vaSpread1.Row = 2
' Define cell as type DATE
vaSpread1.CellType = SS_CELL_TYPE_DATE
vaSpread1.TypeDateCentury = False
vaSpread1.TypeHAlign = SS_CELL_H_ALIGN_LEFT
vaSpread1.TypeSpin = False
vaSpread1.TypeDateFormat = SS_CELL_DATE_FORMAT_MMDDYY
vaSpread1.TypeDateMin = "01011990"
vaSpread1.TypeDateMax = "01011995"
```

[Print](#)

[Copy](#)

[Close](#)

### Example—CellType 1

The following example sets a cell to be an edit cell.

C++

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Define cell as type EDIT
vaSpread1->SetCellType(SS_CELL_TYPE_EDIT);
vaSpread1->SetTypeHAlign(SS_CELL_H_ALIGN_LEFT);
vaSpread1->SetTypeEditCharCase(SS_CELL_EDIT_CASE_NO_CASE);
vaSpread1->SetTypeEditMultiLine(FALSE);
vaSpread1->SetTypeEditLen(60);
```

Visual Basic

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Define cell as type EDIT
vaSpread1.CellType = SS_CELL_TYPE_EDIT
vaSpread1.TypeHAlign = SS_CELL_H_ALIGN_LEFT
vaSpread1.TypeEditCharCase = SS_CELL_EDIT_CASE_NO_CASE
vaSpread1.TypeEditMultiLine = False
vaSpread1.TypeEditLen = 60
```

[Print](#)

[Copy](#)

[Close](#)

### Example—CellType 2

The following example sets a cell to be a float cell.

C++

```
// Select a cell
vaSpread1->SetCol(2);
vaSpread1->SetRow(2);
// Define cell as type FLOAT
vaSpread1->SetCellType(SS_CELL_TYPE_FLOAT);
vaSpread1->SetTypeFloatDecimalPlaces(2);
vaSpread1->SetTypeFloatMin("-9999999.99");
vaSpread1->SetTypeFloatMax("9999999.99");
vaSpread1->SetTypeFloatMoney(FALSE);
vaSpread1->SetTypeFloatSeparator(FALSE);
vaSpread1->SetTypeHAlign(SS_CELL_H_ALIGN_RIGHT);
vaSpread1->SetTypeFloatCurrencyChar('$');
vaSpread1->SetTypeFloatDecimalChar('.');
vaSpread1->SetTypeFloatSepChar(',');
```

Visual Basic

```
' Select a cell
vaSpread1.Col = 2
vaSpread1.Row = 2
' Define cell as type FLOAT
vaSpread1.CellType = SS_CELL_TYPE_FLOAT
vaSpread1.TypeFloatDecimalPlaces = 2
vaSpread1.TypeFloatMin = "-9999999.99"
vaSpread1.TypeFloatMax = "9999999.99"
vaSpread1.TypeFloatMoney = False
vaSpread1.TypeFloatSeparator = False
vaSpread1.TypeHAlign = SS_CELL_H_ALIGN_RIGHT
vaSpread1.TypeFloatCurrencyChar = ASC("$")
vaSpread1.TypeFloatDecimalChar = ASC(".")
vaSpread1.TypeFloatSepChar = ASC(",")
```

[Print](#)

[Copy](#)

[Close](#)

### Example—CellType 3

The following example sets a cell to be an integer cell.

C++

```
// Select a cell
vaSpread1->SetCol(2);
vaSpread1->SetRow(2);
// Define cell as type INTEGER
vaSpread1->SetCellType(SS_CELL_TYPE_INTEGER);
vaSpread1->SetTypeIntegerMin(-99999999);
vaSpread1->SetTypeIntegerMax(99999999);
vaSpread1->SetTypeHAlign(SS_CELL_H_ALIGN_RIGHT);
vaSpread1->SetTypeSpin(FALSE);
vaSpread1->SetTypeIntegerSpinInc(1);
vaSpread1->SetTypeIntegerSpinWrap(FALSE);
```

Visual Basic

```
' Select a cell
vaSpread1.Col = 2
vaSpread1.Row = 2
' Define cell as type INTEGER
vaSpread1.CellType = SS_CELL_TYPE_INTEGER
vaSpread1.TypeIntegerMin = -99999999
vaSpread1.TypeIntegerMax = 99999999
vaSpread1.TypeHAlign = SS_CELL_H_ALIGN_RIGHT
vaSpread1.TypeSpin = False
vaSpread1.TypeIntegerSpinInc = 1
vaSpread1.TypeIntegerSpinWrap = False
```

[Print](#)

[Copy](#)

[Close](#)

#### **Example—CellType 4**

The following example sets a cell to be a PIC cell.

C++

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Define cell as type PIC
vaSpread1->SetCellType(SS_CELL_TYPE_PIC);
vaSpread1->SetTypePicDefaultText("");
vaSpread1->SetTypePicMask("(999)-UUU");
vaSpread1->SetTypeHAlign(SS_CELL_H_ALIGN_LEFT);
```

Visual Basic

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Define cell as type PIC
vaSpread1.CellType = SS_CELL_TYPE_PIC
vaSpread1.TypePicDefaultText = ""
vaSpread1.TypePicMask = "(999)-UUU"
vaSpread1.TypeHAlign = SS_CELL_H_ALIGN_LEFT
```

[Print](#)

[Copy](#)

[Close](#)

### Example—CellType 5

The following example sets a cell to be a static text cell.

C++

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Define cell as type STATIC
vaSpread1->SetCellType(SS_CELL_TYPE_STATIC_TEXT);
vaSpread1->SetTypeHAlign(SS_CELL_H_ALIGN_LEFT);
vaSpread1->SetTypeTextAlignVert(SS_CELL_STATIC_V_ALIGN_TOP);
vaSpread1->SetTypeTextShadow(FALSE);
vaSpread1->SetTypeTextWordWrap(FALSE);
vaSpread1->SetTypeTextPrefix(FALSE);
vaSpread1->SetTypeTextShadowIn(FALSE);
```

Visual Basic

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Define cell as type STATIC
vaSpread1.CellType = SS_CELL_TYPE_STATIC_TEXT
vaSpread1.TypeHAlign = SS_CELL_H_ALIGN_LEFT
vaSpread1.TypeTextAlignVert = SS_CELL_STATIC_V_ALIGN_TOP
vaSpread1.TypeTextShadow = False
vaSpread1.TypeTextWordWrap = False
vaSpread1.TypeTextPrefix = False
vaSpread1.TypeTextShadowIn = False
```



[Print](#)

[Copy](#)

[Close](#)

### Example—CellType 6

The following example sets a cell to be a time cell.

C++

```
// Select a cell
vaSpread1->SetCol(2);
vaSpread1->SetRow(2);
// Define cell as type TIME
vaSpread1->SetCellType(SS_CELL_TYPE_TIME);
vaSpread1->SetTypeTime24Hour(SS_CELL_TIME_12_HOUR_CLOCK);
vaSpread1->SetTypeTimeSeconds(FALSE);
vaSpread1->SetTypeSpin(FALSE);
vaSpread1->SetTypeHAlign(SS_CELL_H_ALIGN_LEFT);
vaSpread1->SetTypeTimeSeparator(':');
vaSpread1->SetTypeTimeMin("000000");
vaSpread1->SetTypeTimeMax("235959");
```

Visual Basic

```
' Select a cell
vaSpread1.Col = 2
vaSpread1.Row = 2
' Define cell as type TIME
vaSpread1.CellType = SS_CELL_TYPE_TIME
vaSpread1.TypeTime24Hour = SS_CELL_TIME_12_HOUR_CLOCK
vaSpread1.TypeTimeSeconds = False
vaSpread1.TypeSpin = False
vaSpread1.TypeHAlign = SS_CELL_H_ALIGN_LEFT
vaSpread1.TypeTimeSeparator = ASC(":")
vaSpread1.TypeTimeMin = "000000"
vaSpread1.TypeTimeMax = "235959"
```

[Print](#)

[Copy](#)

[Close](#)

### Example—CellType 7

The following example sets a cell to be a button cell.

C++

```
// Select a cell
vaSpread1->SetCol(2);
vaSpread1->SetRow(2);
// Define cell as type BUTTON
vaSpread1->SetCellType(SS_CELL_TYPE_BUTTON);
// light gray, RGB(192,192,192)
vaSpread1->SetTypeButtonColor(0x00C0C0C0);
// Convert hBitmap to hPic for the VBX using C++
PIC picture1;
HPIC hPic1;
HBITMAP hBitmap1;
PIC picture2;
HPIC hPic2;
HBITMAP hBitmap2;
hBitmap1=LoadBitmap(AfxGetInstanceHandle( ),"pict1");
picture1.picData.bmp.hbitmap1=hBitmap1;
picture1.picType=PICTURE_BITMAP;
hPic1=AfxSetPict(NULL,&picture1);
hBitmap2=LoadBitmap(AfxGetInstanceHandle( ),"pict2");
picture2.picData.bmp.hbitmap2=hBitmap2;
picture2.picType=PICTURE_BITMAP;
hPic2=AfxSetPict(NULL,&picture2);
vaSpread1->SetTypeButtonPicture(hPic1);
vaSpread1->SetTypeButtonPictureDown(hPic2);
vaSpread1->SetTypeButtonText("Button");
// black, RGB(0,0,0)
vaSpread1->SetTypeButtonTextColor(0x00000000);
// dark gray, RGB(128,128,128)
vaSpread1->SetTypeButtonDarkColor(0x00808080);
// white, RGB(255,255,255)
vaSpread1->SetTypeButtonLightColor(0x00FFFFFF);
// black, RGB(0,0,0)
vaSpread1->SetTypeButtonBorderColor(0x00000000);
vaSpread1->SetTypeButtonShadowSize(2);
vaSpread1->SetTypeButtonType(SS_CELL_BUTTON_NORMAL);
vaSpread1->SetTypeButtonAlign(SS_CELL_BUTTON_ALIGN_LEFT);
```

Visual Basic

```
' Select a cell
vaSpread1.Col = 2
vaSpread1.Row = 2
' Define cell as type BUTTON
vaSpread1.CellType = SS_CELL_TYPE_BUTTON
' light gray, RGB(192,192,192)
vaSpread1.TypeButtonColor = &H00C0C0C0&
vaSpread1.TypeButtonPicture = LoadPicture("c:\bitmaps\pict1.bmp")
vaSpread1.TypeButtonPictureDown = LoadPicture("c:\bitmaps\pict2.bmp")
vaSpread1.TypeButtonText = "Button"
' black, RGB(0,0,0)
vaSpread1.TypeButtonTextColor = &H00000000&
' dark gray, RGB(128,128,128)
vaSpread1.TypeButtonDarkColor = &H00808080&
' white, RGB(255,255,255)
vaSpread1.TypeButtonLightColor = &H00FFFFFF&
' black, RGB(0,0,0)
vaSpread1.TypeButtonBorderColor = &H00000000&
vaSpread1.TypeButtonShadowSize = 2
vaSpread1.TypeButtonType = SS_CELL_BUTTON_NORMAL
```

```
vaSpread1.TypeButtonAlign = SS_CELL_BUTTON_ALIGN_LEFT
```

[Print](#)

[Copy](#)

[Close](#)

### Example—CellType 8

The following example sets a cell to be a combo box cell.

C++

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Define cell as type COMBOBOX
vaSpread1->SetCellType(SS_CELL_TYPE_COMBOBOX);
vaSpread1->SetTypeComboBoxList("Item 1\tItem 2\tItem 3");
vaSpread1->SetTypeComboBoxEditable(FALSE);
```

Visual Basic

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Define cell as type COMBOBOX
vaSpread1.CellType = SS_CELL_TYPE_COMBOBOX
vaSpread1.TypeComboBoxList = "Item 1" + Chr$(9) + "Item 2" + Chr$(9) + "Item 3"
vaSpread1.TypeComboBoxEditable = False
```

[Print](#)

[Copy](#)

[Close](#)

### Example—CellType 9

The following example sets a cell to be a picture cell.

C++

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Define cell as type PICTURE
vaSpread1->SetCellType(SS_CELL_TYPE_PICTURE);
vaSpread1->SetTypePictCenter(FALSE);
vaSpread1->SetTypePictMaintainScale(FALSE);
vaSpread1->SetTypePictStretch(FALSE);
```

Visual Basic

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Define cell as type PICTURE
vaSpread1.CellType = SS_CELL_TYPE_PICTURE
vaSpread1.TypePictCenter = False
vaSpread1.TypePictMaintainScale = False
vaSpread1.TypePictStretch = False
```

[Print](#)

[Copy](#)

[Close](#)

### Example—CellType 10

The following example sets a cell to be a check box cell.

C++

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Define cell as type CHECKBOX
vaSpread1->SetCellType(SS_CELL_TYPE_CHECKBOX);
vaSpread1->SetTypeCheckText("Hello");
vaSpread1->SetTypeCheckCenter(FALSE);
```

Visual Basic

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Define cell as type CHECKBOX
vaSpread1.CellType = SS_CELL_TYPE_CHECKBOX
vaSpread1.TypeCheckText = "Hello"
vaSpread1.TypeCheckCenter = False
```

[Print](#)

[Copy](#)

[Close](#)

### **Example—CellType 11**

The following example sets a cell to be an owner-drawn cell.

C++

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Define cell as type OWNERDRAWN
vaSpread1->SetCellType(SS_CELL_TYPE_OWNER_DRAWN);
```

Visual Basic

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Define cell as type OWNERDRAWN
vaSpread1.CellType = SS_CELL_TYPE_OWNER_DRAWN
```

**See Also**

[BlockMode](#), [Col](#), [Col2](#), [Row](#), [Row2](#) properties

[DataColConfig](#) event



SD	RD	WR	RT	DT
	✓	✓	✓	

## ChangeMade Property

[Example](#)

### Description

Sets or returns whether the user has made changes to data in the spreadsheet. This property is available at run time only.

### Syntax

```
C++          BOOL CSpreadSheet::GetChangeMade( );  
             void CSpreadSheet::SetChangeMade(BOOL value);  
  
Visual Basic  [form.]Spread.ChangeMade[ = boolean%]
```

### Remarks

The default value for the ChangeMade property is False.

The ChangeMade property is automatically set to True whenever the user types new data into the spreadsheet. Set the ChangeMade property to False at any point to determine if the user makes changes to the spreadsheet after that point.

**Note** The ChangeMade property is not automatically set to True if the data is entered in code at run time.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example configures the spreadsheet to beep and display a warning message if the user has made any changes.

**C++**

```
if(vaSpread1->GetChangeMade( ))
{
    MessageBeep(0);
    MessageBox(NULL,"Save before exiting", "Notice", MB_OK);
}
```

**Visual Basic**

```
If vaSpread1.ChangeMade Then
    Beep
    MsgBox "Save before exiting", 48, "Notice"
End If
```

<b>SD</b>	<b>RD</b>	<b>WR</b>	<b>RT</b>	<b>DT</b>
	✓	✓	✓	

## Clip Property

[See Also](#)    [Example](#)

### Description

Sets or returns formatted data into or from a column, a row, or a block of cells. This property is available at run time only.

### Syntax

C++ (OCX)	<b>CString</b> CSpreadSheet::GetClip( ); <b>void</b> CSpreadSheet::SetClip( <b>LPCWSTR</b> value);
C++ (MFC VBX)	<b>CString</b> CSpreadSheet::GetClip( ); <b>void</b> CSpreadSheet::SetClip( <b>LPCSTR</b> value);
C++ (OWL VBX)	<b>string</b> CSpreadSheet::GetClip( ); <b>void</b> CSpreadSheet::SetClip( <b>LPCSTR</b> value);
Visual Basic	[form.]Spread.Clip[ = text\$]

### Remarks

You must set the [Col](#), [Row](#), [Col2](#), and [Row2](#) properties for the target column, row, or block before setting the Clip property. Data for each column is separated by a tab character (ASCII 9, or '\t' in C++); each row is separated by a carriage return/linefeed character (ASCII 13 and ASCII 10, or '\r' and '\n' in C++). You do not need to set the [BlockMode](#) property.

The Clip property works like the [Text](#) property, but is used for blocks of cells. Use the Clip and [ClipValue](#) properties rather than the Text and [Value](#) properties for large amounts of data.

### Data Type

String

[Print](#)

[Copy](#)

[Close](#)

The following example clips a block of text into the spreadsheet.

C++

```
CString buf;
// Specify clipping block
vaSpread1->SetCol(1);
vaSpread1->SetCol2(2);
vaSpread1->SetRow(1);
vaSpread1->SetRow2(2);
// Clip data into the spreadsheet
vaSpread1->SetClip("Text1\tText2\nText3\tText4");
// Clip data out of the spreadsheet
buf = vaSpread1->GetClip( );
```

Visual Basic

```
Dim buf As String
' Specify clipping block
vaSpread1.Col = 1
vaSpread1.Col2 = 2
vaSpread1.Row = 1
vaSpread1.Row2 = 2
' Clip data into the spreadsheet
vaSpread1.Clip = "Text1" + Chr$(9) + "Text2" + Chr$(13) + "Text3" + Chr$(9) +
"Text4"
' Clip data out of the spreadsheet
buf = vaSpread1.Clip
```

**See Also**

[ClipValue](#), [Col](#), [Col2](#), [Row](#), [Row2](#), [Text](#), [Value](#) properties

[GetText](#), [SetText](#) functions

<b>SD</b>	<b>RD</b>	<b>WR</b>	<b>RT</b>	<b>DT</b>
	✓	✓	✓	

## ClipValue Property

[See Also](#)

### Description

Sets or returns unformatted data into or from a column, a row, or a block of cells. This property is available at run time only.

### Syntax

C++ (OCX)	<b>CString</b> CSpreadSheet::GetClipValue( ); <b>void</b> CSpreadSheet::SetClipValue( <b>LPCTSTR</b> value);
C++ (MFC VBX)	<b>CString</b> CSpreadSheet::GetClipValue( ); <b>void</b> CSpreadSheet::SetClipValue( <b>LPCSTR</b> value);
C++ (OWL VBX)	<b>string</b> CSpreadSheet::GetClipValue( ); <b>void</b> CSpreadSheet::SetClipValue( <b>LPCSTR</b> value);
Visual Basic	[form.]Spread.ClipValue[ = text\$]

### Remarks

You must set the [Col](#), [Row](#), [Col2](#), and [Row2](#) properties for the target column, row, or block before using the ClipValue property. Data for each column is separated by a tab character (ASCII 9, or '\t' in C++); each row is separated by a carriage return/linefeed character (ASCII 13 and ASCII 10, or '\r' and '\n' in C++). You do not need to set the [BlockMode](#) property.

The ClipValue property works like the [Value](#) property, but is used for blocks of cells. Use the [Clip](#) and ClipValue properties rather than the [Text](#) and Value properties for large amounts of data. See the [Value](#) property for an explanation of the type of data expected by each cell type.

### Data Type

String

**See Also**

[Clip](#), [Col](#), [Col2](#), [Row](#), [Row2](#), [Text](#), [Value](#) properties

<b>SD</b>	<b>RD</b>	<b>WR</b>	<b>RT</b>	<b>DT</b>
	✓	✓	✓	

## Col Property

[See Also](#)    [Example](#)

### Description

Sets or returns a specific column or specifies the first column of a block of cells on which an operation is to occur. This property is available at run time only.

### Syntax

C++                    **long** CSpreadSheet::GetCol( );  
**void** CSpreadSheet::SetCol(**long** value);

Visual Basic        [form.]Spread.Col[ = value&]

### Remarks

Use the Col property in conjunction with other properties to specify characteristics for a column or a block of cells. Often, the Col property is used with the [Col2](#), [Row](#), and [Row2](#) properties to specify a block of cells to be affected by block properties, such as [Clip](#) or [BlockMode](#).

To specify the cell or cells to be affected by other properties, first specify the necessary cells using the properties listed in the following table, then set the properties that affect the designated cells.

To specify . . .	Set . . .
A cell	Col, Row
A column	Col to non-zero, Row to - 1
A range of columns	Col, Col2
A row	Row to non-zero, Col to - 1
A range of rows	Row, Row2
A block of cells	Col, Row, Col2, Row2
The entire spreadsheet	Col to - 1 or Row to - 1

The Col property has no association with the current active column on the screen.

The Col property is zero-based. Set the Col property to 0 to specify the row header column; set the Col property to - 1 to specify all columns. The default value for the Col property is

-1.

### Data Type

Long Integer



[Print](#)

[Copy](#)

[Close](#)

The following example places the text "Hello" in a cell.

C++

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Enter data into the cell
vaSpread1->SetText("Hello");
```

Visual Basic

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Enter data into the cell
vaSpread1.Text = "Hello"
```

**See Also**

[Action](#), [BlockMode](#), [Clip](#), [Col2](#), [Row](#), [Row2](#) properties

<b>SD</b>	<b>RD</b>	<b>WR</b>	<b>RT</b>	<b>DT</b>
	✓	✓	✓	

## Col2 Property

[See Also](#)    [Example](#)

### Description

Sets or returns the last column of a block of cells on which an operation is to occur. This property is available at run time only.

### Syntax

```
C++          long CSpreadSheet::GetCol2( );
             void CSpreadSheet::SetCol2(long value);

Visual Basic  [form.]Spread.Col2[ = value&]
```

### Remarks

Use the Col2 property in conjunction with other properties to specify characteristics for a column or a block of cells. Often, the Col2 property is used with the [Col](#), [Row](#), and [Row2](#) properties to specify a block of cells to be affected by block properties, such as [Clip](#) or [BlockMode](#).

To specify the cell or cells to be affected by other properties, first specify the necessary cells using the properties listed in the following table, then set the properties that affect the designated cells.

To specify . . .	Set . . .
A cell	Col, Row
A column	Col to non-zero, Row to —1
A range of columns	Col, Col2
A row	Row to non-zero, Col to —1
A range of rows	Row, Row2
A block of cells	Col, Row, Col2, Row2
The entire spreadsheet	Col to —1 or Row to —1

The default value for the Col2 property is 0.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example selects a block of cells and clears the data and format of the cells.

C++

```
// Select a block of cells
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
vaSpread1->SetRow2(4);
vaSpread1->SetCol2(3);
vaSpread1->SetBlockMode(TRUE);
// Clear the data and format of the cells
vaSpread1->SetAction(SS_ACTION_CLEAR);
// Turn block mode off
vaSpread1->SetBlockMode(FALSE);
```

Visual Basic

```
' Select a block of cells
vaSpread1.Row = 2
vaSpread1.Col = 2
vaSpread1.Row2 = 4
vaSpread1.Col2 = 3
vaSpread1.BlockMode = True
' Clear the data and format of the cells
vaSpread1.Action = SS_ACTION_CLEAR
' Turn block mode off
vaSpread1.BlockMode = False
```

**See Also**

[Action](#), [BlockMode](#), [Clip](#), [Col](#), [Row](#), [Row2](#) properties

## ColHeaderDisplay Property

[See Also](#)   [Example](#)

### Description

Sets or returns whether the column header row displays letters or numbers or is blank.

### Syntax

C++ (OCX)      **long** CSpreadSheet::GetColHeaderDisplay( );  
                  **void** CSpreadSheet::SetColHeaderDisplay(**long** value);

C++ (VBX)      **short** CSpreadSheet::GetColHeaderDisplay( );  
                  **void** CSpreadSheet::SetColHeaderDisplay(**short** value);

Visual Basic    [form.]Spread.ColHeaderDisplay[ = setting%]

### OCX Use

Style drop-down list box under Col Headers on the [Headers](#) property page

### Remarks

The following settings are available:

Setting	Description	Constant
0 - Blank	Displays blanks in the column header row	SS_HEADER_BLANK
1 - Display Numbers	Displays numbers in the column header row	SS_HEADER_NUMBERS
2 - Display Letters	(Default) Displays letters in the column header row	SS_HEADER_LETTERS

**Note** Use the ColHeaderDisplay property with column header cells that do not contain data. If you place data into a column header cell using the [Text](#) or [Value](#) properties, the value of the Text or Value property overrides any previously applied ColHeaderDisplay property setting.

### Data Type

Integer (Enumerated)

[Print](#)

[Copy](#)

[Close](#)

The following example displays numbers in the spreadsheet column header row.

C++

```
vaSpread1->SetColHeaderDisplay(SS_HEADER_NUMBERS);
```

Visual Basic

```
vaSpread1.ColHeaderDisplay = SS_HEADER_NUMBERS
```

**See Also**

[RowHeaderDisplay](#), [Text](#), [Value](#) properties



---

## ColHidden Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether a column is hidden. This property is available at run time only.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetColHidden( ); <b>void</b> CSpreadSheet::SetColHidden( <b>BOOL</b> value);
Visual Basic	[form.]Spread.ColHidden[ = boolean%]

### Remarks

Before using the ColHidden property, set the [Col](#) property to specify which column to hide.

The default value for the ColHidden property is False.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example hides spreadsheet column 2.

C++

```
vaSpread1->SetCol(2);  
vaSpread1->SetColHidden(TRUE);
```

Visual Basic

```
vaSpread1.Col = 2  
vaSpread1.ColHidden = True
```

**See Also**

[Col](#), [RowHidden](#) properties

---

## ColPageBreak Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether a page break occurs at the specified column when printing the spreadsheet. This property is available at run time only.

### Syntax

C++                    **BOOL** CSpreadSheet::GetColPageBreak( );  
                         **void** CSpreadSheet::SetColPageBreak(**BOOL** value);

Visual Basic        [form.]Spread.ColPageBreak[ = *boolean%*]

### Remarks

You must set the [Col](#) property to specify the column number before setting the ColPageBreak property. Page breaks occur to the left of the specified column.

The default value for the ColPageBreak property is False.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example sets a print page break before spreadsheet column 2.

C++

```
vaSpread1->SetCol(2);  
vaSpread1->SetColPageBreak(TRUE);
```

Visual Basic

```
vaSpread1.Col = 2  
vaSpread1.ColPageBreak = True
```

**See Also**

[Col](#), [RowPageBreak](#) properties

---

## ColsFrozen Property

[See Also](#)

[Example](#)

### Description

Sets or returns the number of nonscrolling, frozen columns.

### Syntax

C++	<b>long</b> CSpreadSheet::GetColsFrozen( ); <b>void</b> CSpreadSheet::SetColsFrozen( <b>long</b> value);
Visual Basic	[form.]Spread.ColsFrozen[ = value&]

### OCX Use

Columns Frozen for Scrolling box under Col Headers on the [Headers](#) property page

### Remarks

The default value for the ColsFrozen property is 0, which means that only the row header column is frozen.

The ColsFrozen property specifies which columns in the spreadsheet do not scroll. The frozen columns are always the far left  $n$  columns, where  $n$  is the value of this property.

**Note** Setting the ColsFrozen property to a value other than 0 may change the [LeftCol](#) property setting.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example makes the first two columns nonscrolling columns.

C++

```
// Set number of nonscrolling columns  
vaSpread1->SetColsFrozen(2);
```

Visual Basic

```
' Set number of nonscrolling columns  
vaSpread1.ColsFrozen = 2
```



**See Also**

[LeftCol](#), [RowsFrozen](#) properties

---

## ColWidth Property

[See Also](#)   [Example](#)

### Description

Sets or returns the width of the specified column. This property is available at run time only.

### Syntax

C++ (OCX)	<b>double</b> CSpreadSheet::GetColWidth( <b>short</b> <i>Index</i> ); <b>void</b> CSpreadSheet::SetColWidth( <b>long</b> <i>Index</i> , <b>double</b> <i>value</i> );
C++ (VBX)	<b>float</b> CSpreadSheet::GetColWidth( <b>short</b> <i>Index</i> ); <b>void</b> CSpreadSheet::SetColWidth( <b>short</b> <i>Index</i> , <b>float</b> <i>value</i> );
Visual Basic	[ <i>form.</i> ]Spread.ColWidth( <i>Index</i> )[ = <i>value</i> !]

### Remarks

Specify the column using the *Index* parameter. Index values are zero-based; the index value 0 specifies the row header column. If the index is `-1`, the [Col](#) property setting is used.

The units used for the ColWidth property are specified by the [UnitType](#) property. For example, if the UnitType property is set to 2 (Twips), the width must be specified in twips.

**Note** The value set or returned by the ColWidth property does not include the width of the adjacent grid line.

### Data Type

VBX: Single Array  
OCX: Double

[Print](#)

[Copy](#)

[Close](#)

The following example sets the width of spreadsheet column 2 to 12.34 units. The width is specified in characters.

C++

```
// Set the width of a selected column  
vaSpread1->SetColWidth(2,12.34);
```

Visual Basic

```
' Set the width of a selected column  
vaSpread1.ColWidth(2) = 12.34
```

**See Also**

[RowHeight](#), [UnitType](#) properties

[ColWidthChange](#) event

[ColWidthToTwips](#) function

SD	RD	WR	RT	DT
	✓	✓	✓	

## CursorIcon Property

[See Also](#)

[Example](#)

### Description

Sets or returns the path and filename of the icon to display as the mouse pointer. This property is available at run time only. This property is for OCX controls only.

### Syntax

C++ (OCX)                    **HPIC** CSpreadSheet::GetCursorIcon( );  
                                  **void** CSpreadSheet::SetCursorIcon(**HPIC** value);

Visual Basic (OCX)        [form.]Spread.CursorIcon[ = picture]

### Remarks

Use this property to specify the icon to display when the [CursorStyle](#) property is set to 0 (User Defined). You must set the CursorIcon property before setting the CursorStyle property.

### Data Type

Picture

[Print](#)

[Copy](#)

[Close](#)

The following example creates a custom icon for locked cells.

C++

```
// define icon, type, and style
// convert hIcon to hPic for the VBX using C++
Pic picture1;
HPIC hPic1;
HICON hIcon1;
hIcon1=LoadIcon(AfxGetInstanceHandle( ), "secur02b");
picture1.picData.ico.hicon1=hIcon1;
picture1.picType=PICTURE_ICON;
hPic1=AfxSetPict(NULL,&picture1);
vaSpread1->SetCursorIcon(hPic1);
vaSpread1->SetCursorType(SS_CURSOR_TYPE_LOCKEDCELL);
vaSpread1->SetCursorStyle(SS_CURSOR_STYLE_USER_DEFINED);
```

Visual Basic

```
' define icon, type, and style
vaSpread1.CursorIcon = LoadPicture("c:\vb16\icons\misc\secur02b.ico")
vaSpread1.CursorType = SS_CURSOR_TYPE_LOCKEDCELL
vaSpread1.CursorStyle = SS_CURSOR_STYLE_USER_DEFINED
```

**See Also**

[CursorStyle](#), [CursorType](#) properties

## CursorStyle Property

[See Also](#)

[Example](#)

### Description

Sets or returns the style of the mouse pointer. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetCursorStyle( );  
                         **void** CSpreadSheet::SetCursorStyle(**short** value);

Visual Basic        [form.]Spread.CursorStyle[ = setting%]

### Remarks

Use the CursorStyle property to visually indicate the area the mouse pointer is in by setting a different pointer style for different pointer types.

The following settings are available:

Setting	Description	Constant
0 - User Defined (OCX only)	Uses a custom icon specified by the <a href="#">CursorIcon</a> property	SS_CURSOR_STYLE_USER_DEFINED
1 - Default	(Default) Uses the supplied default cursor type	SS_CURSOR_STYLE_DEFAULT
2 - Arrow	Uses the Windows arrow pointer	SS_CURSOR_STYLE_ARROW
3 - Def Col Resize	Uses the default column resize pointer	SS_CURSOR_STYLE_DEFCOLRESIZE
4 - Def Row Resize	Uses the default row resize pointer	SS_CURSOR_STYLE_DEFROWRESIZE

You must set the [CursorType](#) property before setting the CursorStyle property.

If you set the CursorStyle property to 0 (User Defined), you must set the CursorIcon property first.

### Data Type

Integer (Enumerated)



[Print](#)

[Copy](#)

[Close](#)

The following example sets the CursorStyle property to 2 (Windows arrow pointer).

C++

```
// Define pointer type and style
vaSpread1->SetCursorType(SS_CURSOR_TYPE_DEFAULT);
vaSpread1->SetCursorStyle(SS_CURSOR_STYLE_ARROW);
```

Visual Basic

```
' Define pointer type and style
vaSpread1.CursorType = SS_CURSOR_TYPE_DEFAULT
vaSpread1.CursorStyle = SS_CURSOR_STYLE_ARROW
```

**See Also**

[CursorIcon](#), [CursorType](#) properties

## CursorType Property

[See Also](#)

[Example](#)

### Description

Sets or returns the area of the spreadsheet in which to apply the [CursorStyle](#) property. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetCursorType( );  
                         **void** CSpreadSheet::SetCursorType(**short** value);

Visual Basic        [form.]Spread.CursorType[ = setting%]

### Remarks

Once the CursorType property is set to a specific area of the spreadsheet, you can change the style of the pointer for that area with the CursorStyle property.

The following settings are available:

Setting	Description	Constant
0 - Default	(Default) Displays specified pointer in the general area within the spreadsheet	SS_CURSOR_TYPE_DEFAULT
1 - Column Resize	Displays specified pointer in the column resize area	SS_CURSOR_TYPE_COLRESIZE
2 - Row Resize	Displays specified pointer in the row resize area	SS_CURSOR_TYPE_ROWRESIZE
3 - Button	Displays specified pointer in the button when the mouse pointer passes over it	SS_CURSOR_TYPE_BUTTON
4 - Gray Area	Displays specified pointer in the gray area when the mouse pointer passes over it	SS_CURSOR_TYPE_GRAYAREA
5 - Locked Cell	Displays specified pointer within a locked cell when the mouse pointer passes over it	SS_CURSOR_TYPE_LOCKEDCELL
6 - Col Header	Displays specified pointer within a column header when the mouse pointer passes over it	SS_CURSOR_TYPE_COLHEADER
7 - Row Header	Displays specified pointer within a row header when the mouse pointer passes over it	SS_CURSOR_TYPE_ROWHEADER

### Data Type

Integer (Enumerated)

**See Also**

[CursorIcon](#), [CursorStyle](#) properties

---

## DataChanged Property

[See Also](#)

### Description

Returns whether data in a bound Spread control has been changed by some process other than retrieving data from the current record. This property is available at run time only.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetDataChanged( );
Visual Basic	[ <i>form.</i> ]Spread.DataChanged

### OCX Use

The DataChanged property is a standard extender property that can be provided by your container.

### Remarks

The default value for the DataChanged property is False.

If the data in the Spread control changes in a way other than moving to a different record, the DataChanged property is set to True.

Inspect the value of the DataChanged property in your code for the Spread control's [Change](#) event to avoid a cascading event.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

Integer (Boolean)

**See Also**

[Change](#) event

---

## DataColCnt Property

[See Also](#)

[Example](#)

### Description

Returns the last column that contains data in the spreadsheet. This property is available at run time only.

### Syntax

C++                    **long** CSpreadSheet::GetDataColCnt( );

Visual Basic        [form.]Spread.DataColCnt

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example retrieves the last spreadsheet column containing data.

C++

```
x = vaSpread1->GetDataColCnt( );
```

Visual Basic

```
x = vaSpread1.DataColCnt
```



**See Also**

[DataRowCnt](#) property

[EditError](#) event

---

## DataField Property

[See Also](#)    [Example](#)

### Description

Sets or returns the database field name assigned to a column in a bound spreadsheet. This property is available at run time only.

### Syntax

Visual Basic      `[form.]Spread.DataField[ = text$]`

### OCX Use

The DataField property is a standard extender property that can be provided by your container.

### Remarks

The DataField property is valid only when the spreadsheet is bound to a database. Set the [Col](#) property for the column reading the data field before setting the DataField property. If no field names from the database are assigned to the columns, all fields from the database are read into the spreadsheet. If any field names are assigned to specific columns, only those fields are loaded. You do not have to assign field names to contiguous columns.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

For more information about data binding, see [Binding the Spreadsheet to a Database](#).

### Data Type

String

[Print](#)

[Copy](#)

[Close](#)

The following example assigns the database field "Name" to spreadsheet column 2 and retrieves the database field name assigned to column 3.

Visual Basic

```
Dim buf As String
' Set the database field for column 2
vaSpread1.Col = 2
vaSpread1.DataField = "Name"
' Get the database field for column 3
vaSpread1.Col = 3
buf = vaSpread1.DataField
```

**See Also**

[Col](#), [DataFillEvent](#), [DataSource](#), [DAutoCellTypes](#), [DAutoFill](#), [DAutoHeadings](#), [DAutoSave](#), [DAutoSizeCols](#), [DInformActiveRowChange](#) properties

---

## DataFillEvent Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the [DataFill](#) event occurs for a specified column. This property is available at run time only.

### Syntax

Visual Basic      `[form.]Spread.DataFillEvent[ = boolean%]`

### Remarks

The default value for each column is False.

This property is valid only when the spreadsheet is bound to a database. When the DataFillEvent property is set to True for a column, the DataFill event will occur for each cell in that column when the data is read and when the data is written back to the database.

Each column in the spreadsheet can be set independently with this property; therefore, you must use the [Col](#) property to set a specific column before using the DataFillEvent property.

For more information about data binding, see [Binding the Spreadsheet to a Database](#).

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example specifies that the DataFill event can occur for spreadsheet column 1.

Visual Basic

```
vaSpread1.Col = 1  
vaSpread1.DataFillEvent = True
```

**See Also**

[Col](#), [DataField](#), [DAutoCellTypes](#), [DAutoFill](#), [DAutoHeadings](#), [DAutoSave](#), [DAutoSizeCols](#), [DInformActiveRowChange](#) properties

[DataFill](#) event

---

## DataRowCnt Property

[See Also](#)

[Example](#)

### Description

Returns the last row that contains data in the spreadsheet. This property is available at run time only.

### Syntax

C++                    **long** CSpreadSheet::GetDataRowCnt( );

Visual Basic        [form.]Spread.DataRowCnt

### Data Type

Long Integer



[Print](#)

[Copy](#)

[Close](#)

The following example retrieves the last spreadsheet row containing data.

C++

```
x = vaSpread1->GetDataRowCnt ( );
```

Visual Basic

```
x = vaSpread1.DataRowCnt
```

**See Also**

[DataColCnt](#) property

[EditError](#) event

SD	RD	WR	RT	DT
	✓	✓		✓

## DataSource Property

[See Also](#)

### Description

Sets or returns the name of the data control bound to the Spread control. This property is available at design time only. This property is available in Visual Basic only.

### OCX Use

The DataSource property is a standard extender property that can be provided by your container.

### Remarks

To bind a Spread control to a database, you must specify the name of the data control bound to that database using this property.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

For more information about data binding, see [Binding the Spreadsheet to a Database](#).

### Data Type

String

**See Also**

[DataField](#) property

---

## DAutoCellTypes Property

[See Also](#)    [Example](#)

### Description

Sets or returns whether the setting of the cell type for a column receiving data from a database is automatic or manual.

### Syntax

Visual Basic        `[form.]Spread.DAutoCellTypes[ = boolean%]`

### Remarks

The default value for the DAutoCellTypes property is True.

When this property is set to True, the spreadsheet automatically sets the cell type based on the field type for all columns receiving data from the database. When each column is assigned a cell type, the [DataColConfig](#) event occurs, and you can override the assigned cell type for each particular column in the DataColConfig event.

This property is valid only when the spreadsheet is bound to a database. When the DAutoCellTypes property is set to False, you must set the cell type of each column accepting a data field independently of the field data.

For more information about data binding, see [Binding the Spreadsheet to a Database](#).

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example specifies that cell types for bound columns are set automatically based on the database field types.

Visual Basic

```
vaSpread1.DAutoCellTypes = True
```

**See Also**

[DataField](#), [DataFillEvent](#), [DataSource](#), [DAutoFill](#), [DAutoHeadings](#), [DAutoSave](#), [DAutoSizeCols](#), [DInformActiveRowChange](#)  
properties

[DataColConfig](#) event

---

## DAutoFill Property

[See Also](#)    [Example](#)

### Description

Sets or returns whether data is automatically entered from a database into the spreadsheet.

### Syntax

Visual Basic      `[form.]Spread.DAutoFill[ = boolean%]`

### Remarks

The default value for the DAutoFill property is True.

When the DAutoFill property is set to True, the spreadsheet reads the data and places each record into the spreadsheet automatically. When the DAutoFill property is set to False, the application is responsible for loading the data into the spreadsheet.

This property is valid only when the spreadsheet is bound to a database.

For more information about data binding, see [Binding the Spreadsheet to a Database](#).

### Data Type

Integer (Boolean)



[Print](#)

[Copy](#)

[Close](#)

The following example configures the spreadsheet to automatically read data from the database.

Visual Basic

```
vaSpread1.DAutoFill = True
```

**See Also**

[DataField](#), [DataFillEvent](#), [DataSource](#), [DAutoCellTypes](#), [DAutoHeadings](#), [DAutoSave](#), [DAutoSizeCols](#) properties

---

## DAutoHeadings Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether data field names are automatically entered from a database into the spreadsheet column headings.

### Syntax

Visual Basic      `[form.]Spread.DAutoHeadings[ = boolean%]`

### Remarks

The default value for the DAutoHeadings property is True.

When the DAutoHeadings property is set to True, the spreadsheet automatically places the data field names for each column into the column headings. When the DAutoHeadings property is set to False, you must add the column heading for each column to the spreadsheet using the [Text](#) property.

This property is valid only when the spreadsheet is bound to a database.

For more information about data binding, see [Binding the Spreadsheet to a Database](#).

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example configures the spreadsheet to automatically place database field names into the spreadsheet column headings.

Visual Basic

```
vaSpreadl.DAutoHeadings = True
```

**See Also**

[DataField](#), [DataFillEvent](#), [DataSource](#), [DAutoCellTypes](#), [DAutoFill](#), [DAutoSave](#), [DAutoSizeCols](#), [DInformActiveRowChange](#), [Text](#) properties

---

## DAutoSave Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether modified records in a bound spreadsheet are automatically written back to the database.

### Syntax

Visual Basic      `[form.]Spread.DAutoSave[ = boolean%]`

### Remarks

The default value for the DAutoSave property is True.

When the DAutoSave property is set to True, the modified record in the spreadsheet is written back to the database automatically each time the active cell leaves a row in which any data has been modified. When the DAutoSave property is set to False, the modified record does not return to the database, and you must use the [Action](#) property to save the modified record.

This property is valid only when the spreadsheet is bound to a database.

For more information about data binding, see [Binding the Spreadsheet to a Database](#).

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example specifies that modified spreadsheet records are automatically written back to the database.

Visual Basic

```
vaSpread1.DAutoSave = True
```

**See Also**

[Action](#) (15 - Data Save), [DataField](#), [DataFillEvent](#), [DataSource](#), [DAutoCellTypes](#), [DAutoFill](#), [DAutoHeadings](#), [DAutoSizeCols](#), [DInformActiveRowChange](#) properties



## DAutoSizeCols Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether columns in a bound spreadsheet are automatically sized based on the data in the columns.

### Syntax

Visual Basic      `[form.]Spread.DAutoSizeCols[ = setting%]`

### Remarks

The following settings are available:

Setting	Description	Constant
0 - Off	Does not automatically resize spreadsheet columns	SS_AUTOSIZE_NO
1 - Max Col Width	Sets column width to the largest string size	SS_AUTOSIZE_MAX_COL_WIDTH
2 - Best Guess	(Default) Sets column width based on the data type for that field	SS_AUTOSIZE_BEST_GUESS

This property is valid only when the spreadsheet is bound to a database.

When you set the DAutoSizeCols property to 2 (Best Guess), the spreadsheet chooses a size (hard-coded in the Spread software) based on the data type. The actual data might be larger or smaller than the chosen width.

For more information about data binding, see [Binding the Spreadsheet to a Database](#).

### Data Type

Integer (Enumerated)

[Print](#)

[Copy](#)

[Close](#)

The following example specifies that bound spreadsheet columns are automatically resized based on the data in the columns.

Visual Basic

```
vaSpread1.DAutoSizeCols = SS_AUTOSIZE_MAX_COL_WIDTH
```

**See Also**

[DataField](#), [DataFillEvent](#), [DataSource](#), [DAutoCellTypes](#), [DAutoFill](#), [DAutoHeadings](#), [DAutoSave](#), [DInformActiveRowChange](#)  
properties

---

## DestCol Property

[See Also](#)   [Example](#)

### Description

Sets or returns the destination column for an action. This property is available at run time only.

### Syntax

C++	<b>long</b> CSpreadSheet::GetDestCol( ); <b>void</b> CSpreadSheet::SetDestCol( <b>long</b> value);
Visual Basic	[form.]Spread.DestCol[ = value&]

### Remarks

Use this property with the [Action](#) and [DestRow](#) properties to specify the destination of a copy, swap, or move operation.

The default value for the DestCol property is 0.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example copies the data from one block of cells to another. The second (destination) block is specified by the DestCol and DestRow properties and is automatically sized based on the size of the copied block.

C++

```
// Set source location of copy
vaSpread1->SetCol(vaSpread1->GetSelBlockCol( ));
vaSpread1->SetRow(vaSpread1->GetSelBlockRow( ));
vaSpread1->SetCol2(vaSpread1->GetSelBlockCol2( ));
vaSpread1->SetRow2(vaSpread1->GetSelBlockRow2( ));
// Set destination location of copy
NewCol->(1);
NewRow->(1);
vaSpread1->SetDestCol(NewCol);
vaSpread1->SetDestRow(NewRow);
// Copy the data
vaSpread1->SetAction(SS_ACTION_COPY_RANGE);
```

Visual Basic

```
' Set source location of copy
vaSpread1.Col = vaSpread1.SelBlockCol
vaSpread1.Row = vaSpread1.SelBlockRow
vaSpread1.Col2 = vaSpread1.SelBlockCol2
vaSpread1.Row2 = vaSpread1.SelBlockRow2
' Set destination location of copy
NewCol = 1
NewRow = 1
vaSpread1.DestCol = NewCol
vaSpread1.DestRow = NewRow
' Copy the data
vaSpread1.Action = SS_ACTION_COPY_RANGE
```

**See Also**

[Action](#) (19 - Copy Range, 20 - Move Range, 21 - Swap Range), [DestRow](#) properties

---

## DestRow Property

[See Also](#)    [Example](#)

### Description

Sets or returns the destination row for an action. This property is available at run time only.

### Syntax

C++	<b>long</b> CSpreadSheet::GetDestRow( ); <b>void</b> CSpreadSheet::SetDestRow( <b>long</b> value);
Visual Basic	[form.]Spread.DestRow[ = value&]

### Remarks

Use this property with the [Action](#) and [DestCol](#) properties to specify the destination of a copy, swap, or move operation.  
The default value for the DestRow property is 0.

### Data Type

Long Integer

**See Also**

[Action](#) (19 - Copy Range, 20 - Move Range, 21 - Swap Range), [DestCol](#) properties



---

## DInformActiveRowChange Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the spreadsheet informs the data control when the user moves to a new row.

### Syntax

C++                    **BOOL** CSpreadSheet::GetDInformActiveRowChange( );  
                         **void** CSpreadSheet::SetDInformActiveRowChange(**BOOL** value);

Visual Basic        [form.]Spread.DInformActiveRowChange[ = *boolean%*]

### Remarks

The default value for the DInformActiveRowChange property is True.

This property is valid only when the spreadsheet is bound to a database.

For more information about data binding, see [Binding the Spreadsheet to a Database](#).

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example configures the spreadsheet to inform the data control when the user moves to a new row.

C++

```
vaSpread1->SetDInformActiveRowChange(TRUE);
```

Visual Basic

```
vaSpread1.DInformActiveRowChange = True
```

**See Also**

[DataField](#), [DataFillEvent](#), [DataSource](#), [DAutoCellTypes](#), [DAutoFill](#), [DAutoHeadings](#), [DAutoSave](#), [DAutoSizeCols](#) properties

---

## DisplayColHeaders Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the spreadsheet displays the column header row.

### Syntax

C++ **BOOL** CSpreadSheet::GetDisplayColHeaders( );  
**void** CSpreadSheet::SetDisplayColHeaders(**BOOL** value);

Visual Basic [form.]Spread.DisplayColHeaders[ = boolean%]

### OCX Use

Show check box under Col Headers on the [Headers](#) property page

### Remarks

The default value for the DisplayColHeaders property is True.

When the DisplayColHeaders property is set to True, the column header row is displayed. When the DisplayColHeaders property is set to False, the column header row is hidden.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example hides the column header row.

C++

```
vaSpread1->SetDisplayColHeaders (FALSE) ;
```

Visual Basic

```
vaSpread1.DisplayColHeaders = False
```

**See Also**

[DisplayRowHeaders](#) property

---

## DisplayRowHeaders Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the spreadsheet displays the row header column.

### Syntax

C++ **BOOL** CSpreadSheet::GetDisplayRowHeaders( );  
**void** CSpreadSheet::SetDisplayRowHeaders(**BOOL** *value*);

Visual Basic [form.]Spread.DisplayRowHeaders[ = *boolean%*]

### OCX Use

Show check box under Row Headers on the [Headers](#) property page

### Remarks

The default value for the DisplayRowHeaders property is True.

When the DisplayRowHeaders property is set to True, the row header column is displayed. When the DisplayRowHeaders property is set to False, the row header column is hidden.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example hides the row header column.

C++

```
vaSpread1->SetDisplayRowHeaders (FALSE) ;
```

Visual Basic

```
vaSpread1.DisplayRowHeaders = False
```



**See Also**

[DisplayColHeaders](#) property

---

## DragIcon Property

[See Also](#)

### Description

Sets or returns the icon displayed as the mouse pointer when the user drags the Spread control. This property is available in Visual Basic only.

### Syntax

Visual Basic      `[form.]Spread.DragIcon[ = value&]`

### OCX Use

The DragIcon property is a standard extender property that can be provided by your container.

### Remarks

The default value for the DragIcon property is None, meaning the mouse pointer appears as an arrow inside a rectangle when the user drags the Spread control.

You can create a custom pointer by specifying an icon. The file you specify must have the .ICO file-name extension and format.

The DragIcon property is useful for providing visual feedback during a drag-and-drop operation. For example, the icon can indicate that the control is over an appropriate drop target. The pointer changes to the icon specified by the DragIcon property when the user starts to drag the control.

At run time, you can set the DragIcon property to any object's DragIcon or Icon property, or you can assign it an icon by using the LoadPicture function.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

Integer

**See Also**

[DragMode](#) property

## DragMode Property

[See Also](#)

### Description

Sets or returns the dragging mode for drag-and-drop operations. This property is available in Visual Basic only.

### Syntax

Visual Basic      `[form.]Spread.DragMode[ = setting%]`

### OCX Use

The DragMode property is a standard extender property that can be provided by your container.

### Remarks

The following settings are available:

Setting	Description	Constant
0 - Manual	(Default) Requires Drag method to start dragging the control	MANUAL (VB3) vbManual (VB4)
1 - Automatic	Clicking the control automatically starts dragging	AUTOMATIC (VB3) vbAutomatic (VB4)

When the DragMode property is set to 1 (Automatic), the control does not respond as usual to mouse input messages. Use the 0 (Manual) setting to determine when a drag begins or ends; this is useful for initiating dragging in response to a keyboard or menu command, or for allowing a source control to recognize a [MouseDown](#) event prior to dragging.

Releasing the mouse button while the mouse pointer is over a target control or form during a drag operation generates a [DragDrop](#) event for the target object. This ends the drag operation. Dragging may also generate a [DragOver](#) event.

While a control is being dragged, it cannot recognize other user-initiated mouse or keyboard events ([KeyDown](#), [KeyPress](#), [KeyUp](#), [MouseDown](#), [MouseMove](#), or [MouseUp](#)). However, the control can receive events initiated by code or by a DDE link.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

Integer (Enumerated)

**See Also**

[DragIcon](#) property

## EditEnterAction Property

[See Also](#)

[Example](#)

### Description

Sets or returns the action that occurs when the user presses the Enter key.

### Syntax

C++ (OCX)      **long** CSpreadSheet::GetEditEnterAction( );  
                  **void** CSpreadSheet::SetEditEnterAction(**long** value);

C++ (VBX)      **short** CSpreadSheet::GetEditEnterAction( );  
                  **void** CSpreadSheet::SetEditEnterAction(**short** value);

Visual Basic    [form.]Spread.EditEnterAction[ = value%]

### OCX Use

None, Up, Down, Left, Right, Next, Previous, Same, or Next Row option buttons under Enter Key Action on the [Edit Mode](#) property page

### Remarks

The following settings are available:

Setting	Description	Constant
0 - None	(Default) No action occurs	SS_CELL_EDITMODE_EXIT_NONE
1 - Up	Active cell moves up	SS_CELL_EDITMODE_EXIT_UP
2 - Down	Active cell moves down	SS_CELL_EDITMODE_EXIT_DOWN
3 - Left	Active cell moves left	SS_CELL_EDITMODE_EXIT_LEFT
4 - Right	Active cell moves right	SS_CELL_EDITMODE_EXIT_RIGHT
5 - Next	Next cell becomes active (The order is left to right, top to bottom.)	SS_CELL_EDITMODE_EXIT_NEXT
6 - Previous	Previous cell becomes active (The order is right to left, bottom to top.)	SS_CELL_EDITMODE_EXIT_PREVIOUS
7 - Same	Active cell stays on same cell	SS_CELL_EDITMODE_EXIT_SAME
8 - Next Row	Active cell moves to first column of next row	SS_CELL_EDITMODE_EXIT_NEXTROW

### Data Type

Integer (Enumerated)

[Print](#)

[Copy](#)

[Close](#)

The following example specifies that the active cell moves down when the user presses the Enter key.

C++

```
// Define the direction the active cell should take when  
// exiting edit mode  
vaSpread1->SetEditEnterAction(SS_CELL_EDITMODE_EXIT_DOWN);
```

Visual Basic

```
' Define the direction the active cell should take when  
' exiting edit mode  
vaSpread1.EditEnterAction = SS_CELL_EDITMODE_EXIT_DOWN
```

**See Also**

[ProcessTab](#) property



## EditMode Property

[See Also](#)    [Example](#)

### Description

Sets or returns whether a cell is in edit mode. This property is available at run time only.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetEditMode( ); <b>void</b> CSpreadSheet::SetEditMode( <b>BOOL</b> value);
Visual Basic	[form.]Spread.EditMode[ = boolean%]

### Remarks

Edit mode is turned on for a cell when

- the user starts typing in the cell
- the user double-clicks the cell
- the EditMode property is set to True

Edit mode is turned off for a cell when

- the user presses the Enter key
- the user activates a different cell
- the application loses the focus
- the EditMode property is set to False

When edit mode is off, the focus rectangle is visible.

For more information about edit mode, see [Using Edit Mode](#).

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example turns edit mode off.

C++

```
vaSpread1->SetEditMode (FALSE) ;
```

Visual Basic

```
vaSpread1.EditMode = False
```

**See Also**

[EditMode](#) event

---

## EditModePermanent Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether edit mode remains on when moving between cells.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetEditModePermanent( ); <b>void</b> CSpreadSheet::SetEditModePermanent( <b>BOOL</b> value);
Visual Basic	[form.]Spread.EditModePermanent[ = boolean%]

### OCX Use

Permanent check box under Edit Mode on the [Edit Mode](#) property page

### Remarks

The default value for the EditModePermanent property is False.

If the EditModePermanent property is set to True, edit mode remains on when moving between cells. Use this option to emulate a table of edit controls.

**Note** Setting the EditModePermanent property to True automatically sets the [SelectBlockOptions](#) property to 0. If you then set the EditModePermanent property to False, you must reset the SelectBlockOptions property to its original setting.

For more information about edit mode, see [Using Edit Mode](#).

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example specifies that edit mode remains on when moving between cells.

C++

```
// Leave edit mode on when moving between cells  
vaSpread1->SetEditModePermanent(TRUE);
```

Visual Basic

```
' Leave edit mode on when moving between cells  
vaSpread1.EditModePermanent = True
```

**See Also**

[EditMode](#), [SelectBlockOptions](#) properties

[EditMode](#) event

[GetFirstValidCell](#) function

---

## EditModeReplace Property

[See Also](#)   [Example](#)

### Description

Sets or returns whether cell text is replaced or appended during edit mode.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetEditModeReplace( ); <b>void</b> CSpreadSheet::SetEditModeReplace( <b>BOOL</b> value);
Visual Basic	[form.]Spread.EditModeReplace[ = boolean%]

### OCX Use

Replace Existing Text check box under Edit Mode on the [Edit Mode](#) property page

### Remarks

The default value for the EditModeReplace property is False.

When the EditModeReplace property is set to True, text within the cell being edited is replaced by new text when the user begins typing characters. When the EditModeReplace property is set to False, the cursor is placed at the end of the text within the cell being edited and new characters are appended to the old text.

For more information about edit mode, see [Using Edit Mode](#).

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example specifies that existing text in a cell is replaced by the new text typed by the user.

C++

```
// Replace existing text when entering edit mode  
vaSpread1->SetEditModeReplace(TRUE);
```

Visual Basic

```
' Replace existing text when entering edit mode  
vaSpread1.EditModeReplace = True
```



**See Also**

[EditMode](#) event

[EditMode](#) property

---

## Enabled Property

[Example](#)

### Description

Sets or returns whether a control is enabled.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetEnabled( ); <b>void</b> CSpreadSheet::SetEnabled( <b>BOOL</b> value);
Visual Basic	[form.]Spread.Enabled[ = boolean%]

### OCX Use

The Enabled property is a stock property.

### Remarks

This property allows a Spread control to be either enabled or unavailable at run time.

The default value for the Enabled property is True.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example sets a Spread control to be unavailable at run time.

C++

```
vaSpread1->SetEnabled (FALSE) ;
```

Visual Basic

```
vaSpread1.Enabled = False
```

---

## FileNum Property

[See Also](#)    [Example](#)

### Description

Sets or returns the file number to use when saving or loading data. This property is available at run time only. This property is available for VBX controls in Visual Basic and Microsoft Visual C++ only.

### Syntax

C++ (MFC)	<b>short</b> CSpreadSheet::GetFileNum( ); <b>void</b> CSpreadSheet::SetFileNum( <b>short</b> value);
Visual Basic	[form.]Spread.FileNum[ = value%]

### Remarks

An application must open a file and inform the spreadsheet of the open file number before loading or saving data. The assigned number must coincide with the number used in the Open statement. The file must be opened as a binary file.

OCX control users and Borland VBX control users should use the [LoadFromFile](#), [LoadTabFile](#), [SaveToFile](#), and [SaveTabFile](#) functions or methods.

### Data Type

Integer

[Print](#)

[Copy](#)

[Close](#)

The following example saves the entire spreadsheet.

Visual Basic

```
Open "vaSpread1.dat" For Binary As #1
vaSpread1.FileNum = 1
vaSpread1.Action = SS_ACTION_SAVE_ALL
Close #1
```

**See Also**

[Action](#) (8 - Load, 9 - Save All, 10 - Save Values) property

---

## FloatDefCurrencyChar Property

[See Also](#)

[Example](#)

### Description

Sets or returns the default currency character used by all float cells in the spreadsheet. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetFloatDefCurrencyChar( );  
                         **void** CSpreadSheet::SetFloatDefCurrencyChar(**short** value);

Visual Basic        [form.]Spread.FloatDefCurrencyChar[ = value%]

### Remarks

The default value for the FloatDefCurrencyChar property is ASCII 36 (dollar sign).

You can use the ASCII value of the character to designate the currency character.

To specify the currency character for an individual float cell, use the [TypeFloatCurrencyChar](#) property.

### Data Type

Integer

[Print](#)

[Copy](#)

[Close](#)

The following example designates a dollar sign (\$) as the default currency character, a period as the default decimal point character, and a comma as the default separator character.

C++

```
// Define default currency settings  
vaSpread1->SetFloatDefCurrencyChar('$');  
vaSpread1->SetFloatDefDecimalChar('.');  
vaSpread1->SetFloatDefSepChar(',');
```

Visual Basic

```
' Define default currency settings  
vaSpread1.FloatDefCurrencyChar = ASC("$")  
vaSpread1.FloatDefDecimalChar = ASC(".")  
vaSpread1.FloatDefSepChar = ASC(",")
```



**See Also**

[FloatDefDecimalChar](#), [FloatDefSepChar](#), [TypeFloatCurrencyChar](#), [TypeFloatMoney](#) properties

---

## FloatDefDecimalChar Property

[See Also](#)

[Example](#)

### Description

Sets or returns the default decimal point character used by all float cells in the spreadsheet. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetFloatDefDecimalChar( );  
                         **void** CSpreadSheet::SetFloatDefDecimalChar(**short** value);

Visual Basic        [form.]Spread.FloatDefDecimalChar[ = value%]

### Remarks

The default value for the FloatDefDecimalChar property is ASCII 46 (period).

You can use the ASCII value of the character to designate the decimal point character.

To specify the decimal point character for an individual float cell, use the [TypeFloatDecimalChar](#) property.

### Data Type

Integer

**See Also**

[FloatDefCurrencyChar](#), [FloatDefSepChar](#), [TypeFloatDecimalChar](#) properties

---

## FloatDefSepChar Property

[See Also](#)   [Example](#)

### Description

Sets or returns the default separator character used by all float cells in the spreadsheet. This property is available at run time only.

### Syntax

C++	<b>short</b> CSpreadSheet::GetFloatDefSepChar( ); <b>void</b> CSpreadSheet::SetFloatDefSepChar( <b>short</b> value);
Visual Basic	[form.]Spread.FloatDefSepChar[ = value%]

### Remarks

The default value for the FloatDefSepChar property is ASCII 44 (comma).

You can use the ASCII value of the character to designate the separator character.

To specify the separator character for an individual float cell, use the [TypeFloatSepChar](#) property.

### Data Type

Integer

**See Also**

[FloatDefCurrencyChar](#), [FloatDefDecimalChar](#), [TypeFloatSepChar](#) properties

---

## Font Property

[See Also](#)   [Example](#)

### Description

Sets or returns font characteristics of displayed text. This property is available for OCX controls only.

### OCX Use

Fonts property page. The Font property is a stock property.

### Remarks

The Spread OCX control combines the [FontBold](#), [FontItalic](#), [FontName](#), [FontSize](#), [FontStrikethru](#), and [FontUnderline](#) properties into the Font property. However, the OCX control still supports these properties in existing projects.

The attributes of the Font property default to the following values:

Attribute	Default Value
Bold	False
Italic	False
Name	MS Sans Serif
Size	10
Strikethrough	False
Underline	False

Access the font characteristics on the Fonts property page. If you double-click the Font property in the property browser, the control displays a Fonts dialog for setting font characteristics. You can also set font characteristics in code.

For more information about the Fonts property page, see [Spread Control Property Pages](#).

[Print](#)

[Copy](#)

[Close](#)

The following example sets the font to be bold in a Spread control.

OCX

```
vaSpread1.Font.Bold = True
```

**See Also**

[FontBold](#), [FontItalic](#), [FontName](#), [FontSize](#), [FontStrikethru](#), [FontUnderline](#) properties



---

## FontBold, FontItalic, FontStrikethru, FontUnderline Properties

[See Also](#)

### Description

Set or return the style of the displayed text.

### Syntax

```
C++          BOOL CSpreadSheet::GetFontBold( );  
            void CSpreadSheet::SetFontBold(BOOL value);
```

```
Visual Basic  [form.]Spread.FontBold[ = boolean%]
```

**Note** All these properties use similar syntax.

### OCX Use

FontBold, FontItalic: Font Style drop-down list box on the [Fonts](#) property page

FontStrikethru, FontUnderline: Strikeout and Underline check boxes under Effects on the Fonts property page

FontBold, FontItalic, FontStrikethru, and FontUnderline are special custom properties, which have been combined under the [Font](#) property.

### Remarks

The default value for the FontBold property varies depending on the default font of your development environment. The default value for the FontItalic, FontStrikethru, and FontUnderline properties is False. Setting these properties to True turns on the formatting in that style. Setting these properties to False turns off the formatting.

Available fonts vary according to your system configuration, display devices, and printing devices. Font-related properties can be set only to values for which actual fonts exist.

### Notes

— In general, you should change the [FontName](#) property before you set size and style attributes with the [FontSize](#), FontBold, FontItalic, FontStrikethru, and FontUnderline properties. However, when you set TrueType fonts to smaller than 8 points, you should set the point size with the FontSize property, then set the FontName property, and then set the size again with the FontSize property. The Windows environment uses a different font for TrueType fonts that are smaller than 8 points.

— The method for accessing the Font properties varies depending on your development environment. In some environments, such as Visual Basic 4.0, you must double-click the Font property listed in the Properties window to access the Font dialog for setting font characteristics.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

Integer (Boolean)

**See Also**

[Font](#), [FontName](#), [FontSize](#) properties

## FontName Property

[See Also](#)

### Description

Sets or returns the font used to display text in a cell, a column, a row, a block of cells, or the entire spreadsheet.

### Syntax

```
C++ (MFC VBX) CString CSpreadSheet::GetFontName( );  
void CSpreadSheet::SetFontName(LPCSTR value);  
  
C++ (OWL VBX) string CSpreadSheet::GetFontName( );  
void CSpreadSheet::SetFontName(LPCSTR value);  
  
Visual Basic [form.]Spread.FontName[ = text$]
```

### OCX Use

Select the font you want from the Font combo box on the [Fonts](#) property page. FontName is a special custom property, which is included in the [Font](#) property.

### Remarks

The default value for the FontName property is determined by your system settings.

Available fonts vary according to your system configuration, display devices, and printing devices. Font-related properties can be set only to values for which actual fonts exist.

Use the [Col](#) and [Row](#) properties to specify a cell to which to apply a font before setting the FontName property. To apply a font to a block of cells, use the Col, [Col2](#), Row, [Row2](#), and [BlockMode](#) properties to specify the block of cells before setting the FontName property.

If you use the FontName property to retrieve the font for a block of cells that do not all use the same font, the font used in the active cell is returned.

### Notes

— In general, you should change the FontName property before you set size and style attributes with the [FontSize](#), [FontBold](#), [FontItalic](#), [FontStrikethru](#), and [FontUnderline](#) properties. However, when you set TrueType fonts to smaller than 8 points, you should set the point size with the FontSize property, then set the FontName property, and then set the size again with the FontSize property. The Windows environment uses a different font for TrueType fonts that are smaller than 8 points.

— The method for accessing the Font properties varies depending on your development environment. In some environments, such as Visual Basic 4.0, you must double-click the Font property listed in the Properties window to access the Font dialog for setting font characteristics.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

String

**See Also**

[Font](#), [FontBold](#), [FontItalic](#), [FontSize](#), [FontStrikethru](#), [FontUnderline](#) properties

## FontSize Property

[See Also](#)

### Description

Sets or returns the font size of displayed text in a cell, a column, a row, a block of cells, or the entire spreadsheet.

### Syntax

C++	<b>float</b> CSpreadSheet::GetFontSize( ); <b>void</b> CSpreadSheet::SetFontSize( <b>float</b> value);
Visual Basic	[form.]Spread.FontSize[ = value%]

### OCX Use

Size drop-down combo box on the [Fonts](#) property page. FontSize is a special custom property, which is included in the [Font](#) property.

### Remarks

The default value for the FontSize property is determined by your system settings. To change the default, specify the size of the font in points.

The maximum value for the FontSize property is 2048 points.

Available fonts vary according to your system configuration, display devices, and printing devices. Font-related properties can be set only to values for which actual fonts exist.

Use the [Col](#) and [Row](#) properties to specify a cell to which to apply a font size before setting the FontSize property. To apply a font size to a block of cells, use the [Col](#), [Col2](#), [Row](#), [Row2](#), and [BlockMode](#) properties to specify the block of cells before setting the FontSize property.

If you use the FontSize property to retrieve the font size for a block of cells that do not all use the same font size, the font size used in the active cell is returned.

### Notes

— In general, you should change the [FontName](#) property before you set size and style attributes with the FontSize, [FontBold](#), [FontItalic](#), [FontStrikethru](#), and [FontUnderline](#) properties. However, when you set TrueType fonts to smaller than 8 points, you should set the point size with the FontSize property, then set the FontName property, and then set the size again with the FontSize property. The Windows environment uses a different font for TrueType fonts that are smaller than 8 points.

— The method for accessing the Font properties varies depending on your development environment. In some environments, such as Visual Basic 4.0, you must double-click the Font property listed in the Properties window to access the Font dialog for setting font characteristics.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

Single

**See Also**

[Font](#), [FontBold](#), [FontItalic](#), [FontName](#), [FontStrikethru](#), [FontUnderline](#) properties

## ForeColor Property

[See Also](#)

[Example](#)

### Description

Sets or returns the foreground color of a cell, a column, a row, a block of cells, or the entire spreadsheet. This property is available at run time only.

### Syntax

C++ (OCX)	<b>OLE_COLOR</b> CSpreadSheet::GetForeColor( ); <b>void</b> CSpreadSheet::SetForeColor( <b>OLE_COLOR</b> value);
C++ (VBX)	<b>COLORREF</b> CSpreadSheet::GetForeColor( ); <b>void</b> CSpreadSheet::SetForeColor( <b>COLORREF</b> value);
Visual Basic	[form.]Spread.ForeColor[ = color]

### OCX Use

Select ForeColor from the Property Name drop-down list box on the [Colors](#) property page.

### Remarks

The default value for the ForeColor property is &H00000000& (black).

Use the [Col](#) and [Row](#) properties to specify a cell to which to apply a foreground color before setting the ForeColor property. To apply a foreground color to a block of cells, use the Col, [Col2](#), Row, [Row2](#), and [BlockMode](#) properties to specify the block of cells before setting the ForeColor property.

**Note** If you set the ForeColor property without first setting the Col and Row properties, the color you specify is also applied to the column header row and row header column borders.

If you use the ForeColor property to retrieve the foreground color for a block of cells that do not all have the same foreground color, the foreground color of the active cell is returned.

### Data Type

Color

[Print](#)

[Copy](#)

[Close](#)

The following example sets the foreground color of a cell.

**C++**

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Define the foreground color for the cell
// red, RGB(255,0,0)
vaSpread1->SetForeColor(0x000000FF);
```

**Visual Basic**

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Define the foreground color for the cell
' red, RGB(255,0,0)
vaSpread1.ForeColor = &H000000FF&
```



**See Also**

[BackColor](#), [BlockMode](#), [Col](#), [Col2](#), [Row](#), [Row2](#) properties

## Formula Property

[See Also](#)

[Example 1](#)

[Example 2](#)

### Description

Sets or returns a user-defined formula for a cell. This property is available at run time only.

### Syntax

C++ (OCX) **CString** CSpreadSheet::GetFormula( );  
**void** CSpreadSheet::SetFormula(**LPCSTR** value);

C++ (MFC VBX) **CString** CSpreadSheet::GetFormula( );  
**void** CSpreadSheet::SetFormula(**LPCSTR** value);

C++ (OWL VBX) **string** CSpreadSheet::GetFormula( );  
**void** CSpreadSheet::SetFormula(**LPCSTR** value);

Visual Basic [form.]Spread.Formula[ = text\$]

### Remarks

You must set the [Col](#) and [Row](#) properties to specify a cell before you set the Formula property.

Set the Formula property equivalent to a mathematical expression. The Spread control can use *absolute* or *relative* cell references. In absolute cell references, for example, "A1", the column is always referenced as a letter and the row is always referenced as a number. Relative cell references are references that change when copied to another position to reflect the new row and column positions. No space is allowed between the column and row designators.

You define the cell reference style for the spreadsheet by using the [SetRefStyle](#) function. Note that if you use the SS\_REFSTYLE\_A1 value, the cell in which you place the formula is the cell used for relative addressing purposes.

The following table lists the operations available as math expressions.

Operator	Description	Literal / Literal	Cell Ref / Literal	Cell Ref / Cell Ref
+	Addition	2 + 2	A1 + 2	A1 + B2
—	Subtraction	4 — 2	A1 — 2	A1 — B2
^	Exponentiation	2 ^ 2	A1 ^ 2	A1 ^ B2
*	Multiplication	2 * 2	A1 * 2	A1 * B2
/	Division	2 / 2	A1 / 2	A1 / B2
&	Logical And	2 & 2	A1 & 2	A1 & B2
	Logical Or	2   2	A1   2	A1   B2
:	Creates a range			A1: B2
#	Wild card		A# * 2	A# * B#

**Note** The wild card operator (#) is available only when the reference style is SS\_REFSTYLE\_DEFAULT. For more information, see the [GetRefStyle](#) and [SetRefStyle](#) functions.

The following table lists the functions available as math expressions. The letter A represents a relational expression, for example, A1 > B1. The letters B and C represent references to cell or mathematical expressions that contain cell references. The letter D represents the amount of precision.

Function	Description
ABS(B)	The absolute value of the cell is returned.
ADD(A, B)	Adds the two elements.
IF(A,B,C)	If A is nonzero then B is returned. If A is zero then C is returned. A can contain one of the relational operators: greater than (>), less than (<), equal to (=), or not equal to (!).
ISEMPTY(B)	If B is empty then a 1 is returned. If B is not empty then a zero is returned.
MAX(A,B, . . .)	Returns the maximum value of all arguments. Accepts a variable number of arguments. Each argument can be a cell range, a float value, or an integer value.
MIN(A, B, . . .)	Returns the minimum value of all arguments. Accepts a variable number of arguments. Each argument can be a cell range, a float value, or an integer value.
NEG(B)	Changes the sign of the value. For example, NEG (—15) = 15.
NOT(B)	If B is zero then a 1 is returned. If B is not zero then a zero is returned.

PMT(Amt, Interest, Term, TermsPerYear)	Returns the payment given the specified loan information.
ROUNDUP(B,D)	Rounds the value up to the next whole number, using the specified number of decimal places (D). A zero can be used for no decimal places. Negative precision specifies tenths, hundredths, etc.
ROUND(B,D)	Rounds the value to the nearest whole number, using the specified number of decimal places (D). A zero can be used for no decimal places. Negative precision specifies tenths, hundredths, etc.
SUM(B,C, . . .)	Sums cells or blocks. Accepts a variable number of arguments. Each argument can be a cell range, a float value, or an integer value.
TRUNCATE(B,D)	Rounds the value down to the next whole number, using the specified number of decimal places (D). A zero can be used for no decimal places. Negative precision specifies tenths, hundredths, etc.

**Data Type**

String

[Print](#)

[Copy](#)

[Close](#)

**Example 1**

The following example sums the range of cells A1 through A4 and places the total in cell B2.

C++

```
vaSpread1->SetRow(2);  
vaSpread1->SetCol(2);  
vaSpread1->SetFormula("SUM(A1:A4)");
```

Visual Basic

```
vaSpread1.Row = 2  
vaSpread1.Col = 2  
vaSpread1.Formula = "SUM(A1:A4) "
```

[Print](#)

[Copy](#)

[Close](#)

### Example 2

The following example copies the formula in cell (2,2) to cell (5,5).

C++

```
CString buf;  
vaSpread1->SetRow(2);  
vaSpread1->SetCol(2);  
buf = vaSpread1->GetFormula( );  
vaSpread1->SetRow(5);  
vaSpread1->SetCol(5);  
vaSpread1->SetText(buf);
```

Visual Basic

```
Dim buf As String  
vaSpread1.Row = 2  
vaSpread1.Col = 2  
buf = vaSpread1.Formula  
vaSpread1.Row = 5  
vaSpread1.Col = 5  
vaSpread1.Text = buf
```

**See Also**

[Action](#) (11 - Recalculate), [AllowUserFormulas](#), [AutoCalc](#), [FormulaSync](#) properties

[CustomFunction](#) event

[GetCustomName](#), [GetFormulaSync](#), [GetRefStyle](#), [IsFormulaValid](#), [SetCustomName](#), [SetFormulaSync](#), [SetRefStyle](#) functions

---

## FormulaSync Property

[See Also](#)   [Example](#)

### Description

Sets or returns whether formulas in the spreadsheet that use absolute cell references are adjusted when columns or rows are added or deleted or when blocks of cells are moved or swapped. This property is available for OCX controls only.

### Syntax

C++ (OCX)        **BOOL** CSpreadSheet::GetFormulaSync( );  
                  **void** CSpreadSheet::SetFormulaSync(**BOOL** value);

Visual Basic (OCX) [form.]Spread.FormulaSync[ = boolean%]

### OCX Use

Synchronize Formulas check box under Calculations on the [General](#) property page

### Remarks

The default value for the FormulaSync property is True, which causes the spreadsheet to adjust formulas that use absolute cell references when you insert or delete columns or rows or when you move or swap blocks of cells.

If your formulas use absolute cell references, you must update your formulas manually.

Formulas are not adjusted when you use the [Action](#) property (setting 25 (Sort)) to sort blocks of cells in the spreadsheet.

VBX control users should use the [GetFormulaSync](#) and [SetFormulaSync](#) functions.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example sets the spreadsheet to not update formulas and inserts a row above row 3.

C++

```
// Set the spreadsheet to not update formulas
vaSpread1->SetFormulaSync(FALSE);
// Insert a row above row 3
vaSpread1->SetRow(3);
vaSpread1->SetAction(SS_ACTION_INSERT_ROW);
```

Visual Basic

```
' Set the spreadsheet to not update formulas
vaSpread1.FormulaSync = False
' Insert a row above row 3
vaSpread1.Row = 3
vaSpread1.Action = SS_ACTION_INSERT_ROW
```



**See Also**

[Formula](#) property

[GetFormulaSync](#), [SetFormulaSync](#) functions

---

## GrayAreaBackColor Property

[Example](#)

### Description

Sets or returns the gray area background color of the spreadsheet.

### Syntax

C++ (OCX)	<b>unsigned long</b> CSpreadSheet::GetGrayAreaBackColor( ); <b>void</b> CSpreadSheet::SetGrayAreaBackColor( <b>unsigned long</b> value);
C++ (VBX)	<b>COLORREF</b> CSpreadSheet::GetGrayAreaBackColor( ); <b>void</b> CSpreadSheet::SetGrayAreaBackColor( <b>COLORREF</b> value);
Visual Basic	[form.]Spread.GrayAreaBackColor[ = color]

### OCX Use

Select GrayAreaBackColor from the Property Name drop-down list box on the [Colors](#) property page.

### Remarks

If the user scrolls beyond the maximum rows or columns, a gray area appears, indicating no additional cells exist. Change the color of this area by setting the GrayAreaBackColor property.

The default value for the GrayAreaBackColor property is &H00C0C0C0& (light gray).

### Data Type

VBX: Color  
OCX: Unsigned Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets the gray area background color to red.

C++

```
// Define gray area background color  
// red, RGB(255,0,0)  
vaSpread1->SetGrayAreaBackColor(0x000000FF);
```

Visual Basic

```
' Define gray area background color  
' red, RGB(255,0,0)  
vaSpread1.GrayAreaBackColor = &H000000FF&
```

---

## GridColor Property

[See Also](#)    [Example](#)

### Description

Sets or returns the grid display color.

### Syntax

C++ (OCX)	<b>unsigned long</b> CSpreadSheet::GetGridColor( ); <b>void</b> CSpreadSheet::SetGridColor( <b>unsigned long value</b> );
C++ (VBX)	<b>COLORREF</b> CSpreadSheet::GetGridColor( ); <b>void</b> CSpreadSheet::SetGridColor( <b>COLORREF value</b> );
Visual Basic	[form.]Spread.GridColor[ = color]

### OCX Use

Select GridColor from the Property Name drop-down list box on the [Colors](#) property page.

### Remarks

The default value for the GridColor property is &H00C0C0C0& (light gray).

This property defines the color of the spreadsheet grid lines.

By default, the background color of cells overlaps the bottom and right sides of the surrounding grid lines, hiding those portions of the grid lines from view. To display all sides of the grid lines, set the [BackColorStyle](#) property to 1 (Under Grid).

### Data Type

VBX: Color  
OCX: Unsigned Long Integer

**See Also**

[BackColorStyle](#), [GridShowHoriz](#), [GridShowVert](#), [GridSolid](#) properties

---

## GridShowHoriz Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether to display horizontal grid lines on the spreadsheet.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetGridShowHoriz( ); <b>void</b> CSpreadSheet::SetGridShowHoriz( <b>BOOL</b> value);
Visual Basic	[form.]Spread.GridShowHoriz[ = boolean%]

### OCX Use

Show Horizontal Lines check box under Grid Lines on the [Display](#) property page

### Remarks

The default value for the GridShowHoriz property is True, which displays horizontal grid lines.

By default, the background color of cells overlaps the bottom and right sides of the surrounding grid lines, hiding those portions of the grid lines from view. To display all sides of the grid lines, set the [BackColorStyle](#) property to 1 (Under Grid).

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example creates a grid with solid red horizontal grid lines.

C++

```
// Define grid type, style, and color
vaSpread1->SetGridShowHoriz(TRUE);
vaSpread1->SetGridShowVert(FALSE);
vaSpread1->SetGridSolid(TRUE);
// red, RGB(255,0,0)
vaSpread1->SetGridColor(0x000000FF);
```

Visual Basic

```
' Define grid type, style, and color
vaSpread1.GridShowHoriz = True
vaSpread1.GridShowVert = False
vaSpread1.GridSolid = True
' red, RGB(255,0,0)
vaSpread1.GridColor = &H000000FF&
```

**See Also**

[BackColorStyle](#), [GridColor](#), [GridShowVert](#), [GridSolid](#) properties



---

## GridShowVert Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether to display vertical grid lines on the spreadsheet.

### Syntax

C++                    **BOOL** CSpreadSheet::GetGridShowVert( );  
                         **void** CSpreadSheet::SetGridShowVert(**BOOL** value);

Visual Basic        [form.]Spread.GridShowVert[ = boolean%]

### OCX Use

Show Vertical Lines check box under Grid Lines on the [Display](#) property page

### Remarks

The default value for the GridShowVert property is True, which displays vertical grid lines.

By default, the background color of cells overlaps the bottom and right sides of the surrounding grid lines, hiding those portions of the grid lines from view. To display all sides of the grid lines, set the [BackColorStyle](#) property to 1 (Under Grid).

### Data Type

Integer (Boolean)

**See Also**

[BackColorStyle](#), [GridColor](#), [GridShowHoriz](#), [GridSolid](#) properties

---

## GridSolid Property

[See Also](#)    [Example](#)

### Description

Sets or returns the type of grid lines displayed on the spreadsheet.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetGridSolid( ); <b>void</b> CSpreadSheet::SetGridSolid( <b>BOOL</b> value);
Visual Basic	[form.]Spread.GridSolid[ = boolean%]

### OCX Use

Make Grid Lines Solid check box under Grid Lines on the [Display](#) property page

### Remarks

The default value for the GridSolid property is True.

When the GridSolid property is set to True, the grid lines are displayed as solid lines. When the GridSolid property is set to False, the grid lines are displayed as dotted lines.

You must set the [GridShowHoriz](#) and [GridShowVert](#) properties to True to display grid lines.

By default, the background color of cells overlaps the bottom and right sides of the surrounding grid lines, hiding those portions of the grid lines from view. To display all sides of the grid lines, set the [BackColorStyle](#) property to 1 (Under Grid).

### Data Type

Integer (Boolean)

**See Also**

[BackColorStyle](#), [GridColor](#), [GridShowHoriz](#), [GridShowVert](#) properties

---

## hDCPrinter Property

[See Also](#)

### Description

Sets or returns a device context handle (HDC) for a printer for creating custom printing configurations. This property is available at run time only.

### Syntax

C++ (OCX)	<b>long</b> CSpreadSheet::GethDCPrinter( ); <b>void</b> CSpreadSheet::SethDCPrinter( <b>long</b> value);
C++ (VBX)	<b>short</b> CSpreadSheet::GethDCPrinter( ); <b>void</b> CSpreadSheet::SethDCPrinter( <b>short</b> value);
Visual Basic	[form.]Spread.hDCPrinter[ = value%]

### Remarks

The default setting for the hDCPrinter property is 0, which specifies that the Windows default settings are used to create a device context handle for the printer. The hDCPrinter property can be set to a printer HDC created using a common dialog to provide the user with more control over printing configurations.

### Data Type

VBX: Integer  
OCX: Long Integer

**See Also**

[Action](#) (13 - Print, 32 - Smart Print) property

---

## Height Property

[See Also](#)

### Description

Sets or returns the height of the Spread control.

### Syntax

C++	<b>long</b> CSpreadSheet::GetHeight( ); <b>void</b> CSpreadSheet::SetHeight( <b>long</b> value);
Visual Basic	[form.]Spread.Height[ = value%]

### OCX Use

The Height property is a standard extender property that can be provided by your container.

### Remarks

The value of this property changes as the user or the application changes the size of the control. The maximum value for this property is determined by the system.

The measurement for the height of the control is calculated from the center of the control's border so that controls with different border widths align correctly.

In Visual Basic, the measurement unit used by the Height property depends on the setting of the form's ScaleMode property. The default ScaleMode setting is twips (1/1440 of an inch). C++ uses pixels as the measurement unit.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

Single

**See Also**

[Width](#) property



---

## HelpContextID Property

### Description

Sets or returns an associated context number for the Spread control.

### Syntax

C++	<b>long</b> CSpreadSheet::GetHelpContextID( ); <b>void</b> CSpreadSheet::SetHelpContextID( <b>long</b> value);
Visual Basic	[form.]Spread.HelpContextID[ = value&]

### OCX Use

The HelpContextID property is a standard extender property that can be provided by your container.

### Remarks

Use this property to provide context-sensitive help identifiers for your application. To create context-sensitive help, assign the same context number to both the control and the associated help topic when you compile your help file.

If you have created a Windows environment help file for your application, when the user presses the F1 key, Visual Basic automatically calls the help file and requests the topic identified by the current context number.

The current context number is the value of the HelpContextID property for the control that has the focus. If the value of the HelpContextID property is 0, Visual Basic looks in the HelpContextID of the control's container, and then that object's container, and so on. If Visual Basic cannot find a nonzero current context number, it ignores the F1 key.

**Note** Building a help file requires the Microsoft Windows Help Compiler.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

Long Integer

---

## hWnd Property

### Description

Returns the window handle (HWND) of the Spread control. This property is available at run time only.

### Syntax

C++ (OCX)	<b>OLE_HANDLE</b> CSpreadSheet::GethWnd( );
C++ (VBX)	<b>short</b> CSpreadSheet::GethWnd( );
Visual Basic	[ <i>form.</i> ]Spread(hWnd)

### OCX Use

The hWnd property is a stock property.

### Remarks

The Windows environment identifies each form and control in an application by assigning it a handle or HWND. Use the hWnd property with Windows API calls.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

Integer

---

## Index Property

### Description

Returns the number that uniquely identifies a control in a control array.

### Syntax

Visual Basic      `[form.]Spread[(integer)].Index[ = value%]`

### OCX Use

The Index property is a standard extender property that can be provided by your container.

### Remarks

This property is available only when the control is part of an array.

Because control array elements share the same [Name](#) property setting, you must use the Index property at run time to refer to a particular control in the array. Place the index number in parentheses next to the control array name, for example:

```
vaSpread1 (3)
```

**Note** To remove a control from a control array, change the control's Name property setting.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

Integer

---

## IsBlockSelected Property

[See Also](#)

[Example](#)

### Description

Returns whether a block of cells is selected. This property is available at run time only.

### Syntax

C++ **BOOL** CSpreadSheet::GetIsBlockSelected( );

Visual Basic [form.]Spread.IsBlockSelected

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example checks the Spread control for selected blocks and turns the background color of the selected cells to blue.

C++

```
// Determine if a multiple selection exists
vaSpread1->SetAction(SS_ACTION_GET_MULTI_SELECTION);
// Get whether a block of cells is selected
if(vaSpread1->GetIsBlockSelected( ) || vaSpread1->GetMultiSelCount( ))
{
    int i;
    // Set the background color for each cell block to blue
    vaSpread1->SetBlockMode(TRUE);
    for(i = 0; i < vaSpread1->GetMultiSelCount( ); i++)
    {
        vaSpread1->SetMultiSelIndex(i);
        // blue, RGB(0,0,255)
        vaSpread1->SetBackColor(0x00FF0000);
    }
    vaSpread1->SetBlockMode(FALSE);
}
```

Visual Basic

```
' Determine if a multiple selection exists
vaSpread1.Action = SS_ACTION_GET_MULTI_SELECTION
' Get whether a block of cells is selected
If vaSpread1.IsBlockSelected Or vaSpread1.MultiSelCount Then
' Set the background color for each cell block to blue
vaSpread1.BlockMode = True
For x = 0 To vaSpread1.MultiSelCount - 1
    vaSpread1.MultiSelIndex = i
    ' blue, RGB(0,0,255)
    vaSpread1.BackColor = &H00FF0000&
Next x
vaSpread1.BlockMode = False
End If
```

**See Also**

[Action](#) (2 - Select Block, 14 - Deselect Block, 18 - Get MultiSel Blocks) property

[BlockSelected](#) event

---

## Left Property

[See Also](#)

### Description

Sets or returns the distance between the internal left edge of an object and the left edge of its container.

### Syntax

C++ (OCX)	<b>long</b> CSpreadSheet::GetLeft( ); <b>void</b> CSpreadSheet::SetLeft( <b>long</b> value);
C++ (VBX)	<b>short</b> CSpreadSheet::GetLeft( ); <b>void</b> CSpreadSheet::SetLeft( <b>short</b> value);
Visual Basic	[form.]Spread.Left[ = value!]

### OCX Use

The Left property is a standard extender property that can be provided by your container.

### Remarks

The value for this property changes as the user or the application moves the control. The Left property is measured in units according to the coordinate system of its container.

In Visual Basic, the measurement unit used by the Left property depends on the setting of the form's ScaleMode property. The default ScaleMode setting is twips (1/1440 of an inch). C++ uses pixels as the measurement unit.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

VBX: Single  
OCX: Long Integer

**See Also**

[Top](#) property



---

## LeftCol Property

[See Also](#)   [Example](#)

### Description

Sets or returns the number of the far left column in the viewing area. This property is available at run time only.

### Syntax

C++	<b>long</b> CSpreadSheet::GetLeftCol( ); <b>void</b> CSpreadSheet::SetLeftCol( <b>long</b> value);
Visual Basic	[form.]Spread.LeftCol[ = value&]

### Remarks

Use the LeftCol property to set a specific column as the far left column in the viewing area.

Setting the [ColsFrozen](#) property to a value other than 0 may change the LeftCol property setting.

**Note** Changing the LeftCol property does not affect the position of the active cell.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example displays column 5 as the far left column in the viewing area.

C++

```
vaSpread1->SetLeftCol (5);
```

Visual Basic

```
vaSpread1.LeftCol = 5
```

**See Also**

[Action](#) (1 - Go To), [ColsFrozen](#), [TopRow](#) properties

[TopLeftChange](#) event

[GetCellFromScreenCoord](#) function

---

## Lock Property

[See Also](#)   [Example](#)

### Description

Sets or returns whether the user can type data into a cell, a column, a row, a block of cells, or the entire spreadsheet. This property is available at run time only.

### Syntax

C++                    **BOOL** CSpreadSheet::GetLock( );  
                         **void** CSpreadSheet::SetLock(**BOOL** value);

Visual Basic        [form.]Spread.Lock[ = boolean%]

### Remarks

You must perform three steps to lock cells:

1. First, specify the cells you want to prevent the user from editing. Use the [Col](#) and [Row](#) properties to specify a cell to mark as locked. To mark a block of cells as locked, use the Col, [Col2](#), Row, [Row2](#), and [BlockMode](#) properties to specify the block of cells before setting the Lock property.
2. Next, mark the cells as locked by setting the Lock property.
3. Complete locking the cells by setting the [Protect](#) property to True.

Note that when the Protect property is set to False, the user can still edit cells marked as locked with the Lock property. By default, the Protect property is set to True.

The default value for the Lock property is False.

If you use the Lock property to retrieve the lock status for a block of cells that do not all have the same lock status, the lock status of the active cell is returned.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example locks row 7 of the spreadsheet.

C++

```
// Select cells
vaSpread1->SetRow(7);
vaSpread1->SetCol(-1);
// Lock cells
vaSpread1->SetLock(TRUE);
```

Visual Basic

```
' Select cells
vaSpread1.Row = 7
vaSpread1.Col = -1
' Lock cells
vaSpread1.Lock = True
```

**See Also**

[BlockMode](#), [Col](#), [Col2](#), [LockBackColor](#), [LockForeColor](#), [Protect](#), [Row](#), [Row2](#) properties

[EditError](#) event

[GetFirstValidCell](#) function

## LockBackColor Property

[See Also](#)

[Example](#)

### Description

Sets or returns the background color of locked cells. This property is available at run time only.

### Syntax

C++ (OCX)	<b>unsigned long</b> CSpreadSheet::GetLockBackColor( ); <b>void</b> CSpreadSheet::SetLockBackColor( <b>unsigned long</b> value);
C++ (VBX)	<b>COLORREF</b> CSpreadSheet::GetLockBackColor( ); <b>void</b> CSpreadSheet::SetLockBackColor( <b>COLORREF</b> value);
Visual Basic	[form.]Spread.LockBackColor[ = color]

### OCX Use

Select LockBackColor from the Property Name drop-down list box on the [Colors](#) property page.

### Remarks

The default value for the LockBackColor property is SPREAD\_COLOR\_NONE. When the LockBackColor property is set to this value, the current cell background color set by the [BackColor](#) property is used for the locked cells.

**Note** Once you set the LockBackColor property to a value other than SPREAD\_COLOR\_NONE, you cannot reset the property to this value.

### Data Type

VBX: Color

OCX: Unsigned Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets the background color for locked cells to red.

C++

```
// Define the background locked cell color
// red, RGB(255,0,0)
vaSpread1->SetLockBackColor(0x000000FF);
```

Visual Basic

```
' Define the background locked cell color
' red, RGB(255,0,0)
vaSpread1.LockBackColor = &H000000FF&
```



**See Also**

[BackColor](#), [Lock](#), [LockForeColor](#), [Protect](#) properties

## LockForeColor Property

[See Also](#)

[Example](#)

### Description

Sets or returns the foreground color of locked cells. This property is available at run time only.

### Syntax

C++ (OCX)	<b>unsigned long</b> CSpreadSheet::GetLockForeColor( ); <b>void</b> CSpreadSheet::SetLockForeColor( <b>unsigned long</b> value);
C++ (VBX)	<b>COLORREF</b> CSpreadSheet::GetLockForeColor( ); <b>void</b> CSpreadSheet::SetLockForeColor( <b>COLORREF</b> value);
Visual Basic	[form.]Spread.LockForeColor[ = color]

### OCX Use

Select LockForeColor from the Property Name drop-down list box on the [Colors](#) property page.

### Remarks

The default value for the LockForeColor property is SPREAD\_COLOR\_NONE. When the LockForeColor property is set to this value, the current cell foreground color set by the [ForeColor](#) property is used for the locked cells.

**Note** Once you set the LockForeColor property to a value other than SPREAD\_COLOR\_NONE, you cannot reset the property to this value.

### Data Type

VBX: Color

OCX: Unsigned Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets the foreground color for locked cells to blue.

C++

```
// Define the foreground locked cell color
// blue, RGB(0,0,255)
vaSpread1->SetLockForeColor(0x00FF0000);
```

Visual Basic

```
' Define the foreground locked cell color
' blue, RGB(0,0,255)
vaSpread1.LockForeColor = &H00FF0000&
```

**See Also**

[ForeColor](#), [Lock](#), [LockBackColor](#), [Protect](#) properties

---

## MaxCols Property

[See Also](#)    [Example](#)

### Description

Sets or returns the maximum number of usable columns.

### Syntax

C++	<b>long</b> CSpreadSheet::GetMaxCols( ); <b>void</b> CSpreadSheet::SetMaxCols( <b>long</b> value);
Visual Basic	[form.]Spread.MaxCols[ = value&]

### OCX Use

Max Number of Columns box under Col Headers on the [Headers](#) property page

### Remarks

The default value for the MaxCols property is 500.

The maximum number of available columns is two billion. A gray area appears to the right of the last usable column.

To increase the maximum number of columns when a column is inserted, first use the MaxCols property to increment the maximum number of columns by one, then use the [Action](#) property to insert the column.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets the maximum number of columns for the spreadsheet to two.

C++

```
vaSpread1->SetMaxCols (2) ;
```

Visual Basic

```
vaSpread1.MaxCols = 2
```

**See Also**

[Action](#) (6 - Insert Col), [DataColCnt](#), [DataRowCnt](#), [MaxRows](#) properties

---

## MaxRows Property

[See Also](#)    [Example](#)

### Description

Sets or returns the maximum number of usable rows.

### Syntax

C++	<b>long</b> CSpreadSheet::GetMaxRows( ); <b>void</b> CSpreadSheet::SetMaxRows( <b>long</b> value);
Visual Basic	[form.]Spread.MaxRows[ = value&]

### OCX Use

Max Number of Rows box under Row Headers on the [Headers](#) property page

### Remarks

The default value for the MaxRows property is 500.

The maximum number of available rows is two billion. A gray area appears below the last usable row.

To increase the maximum number of rows when a row is inserted, first use the MaxRows property to increment the maximum number of rows by one, then use the [Action](#) property to insert the row.

### Data Type

Long Integer



[Print](#)

[Copy](#)

[Close](#)

The following example sets the maximum number of rows for the spreadsheet to two.

C++

```
vaSpread1->SetMaxRows (2) ;
```

Visual Basic

```
vaSpread1.MaxRows = 2
```

**See Also**

[Action](#) (7 - Insert Row), [DataColCnt](#), [DataRowCnt](#), [MaxCols](#) properties

---

## MaxTextCellHeight Property

[See Also](#)    [Example](#)

### Description

Returns the minimum height of the specified cell necessary to display all the text within the cell. This property is available at run time only.

### Syntax

C++ (OCX)	<b>double</b> CSpreadSheet::GetMaxTextCellHeight( );
C++ (VBX)	<b>short</b> CSpreadSheet::GetMaxTextCellHeight( );
Visual Basic	[ <i>form.</i> ]Spread.MaxTextCellHeight

### Remarks

Use the [Col](#) and [Row](#) properties to specify the cell.

For edit and static text cells, the value returned varies depending on the setting of the [TypeEditMultiLine](#) property (for edit cells) or the [TypeTextWordWrap](#) property (for static text cells), as listed in the following table.

TypeEditMultiLine or TypeTextWordWrap property setting	Value returned
True	The minimum row height necessary to display all the text in the cell given the current column width and font size
False	The minimum row height necessary to display one line of text given the current font size

### Data Type

VBX: Single  
OCX: Double

[Print](#)

[Copy](#)

[Close](#)

The following example determines the maximum height of a cell and assigns this value as the row height using the [RowHeight](#) property.

C++

```
// Size row height to fill max height of the text in the cell
vaSpread1->SetCol(1);
vaSpread1->SetRow(1);
x = vaSpread1->GetMaxTextCellHeight( );
vaSpread1->SetRowHeight(1,x);
```

Visual Basic

```
' Size row height to fill max height of the text in the cell
vaSpread1.Col = 1
vaSpread1.Row = 1
x = vaSpread1.MaxTextCellHeight
vaSpread1.RowHeight(1) = x
```

**See Also**

[AutoSize](#), [Col](#), [MaxTextCellWidth](#), [MaxTextColWidth](#), [MaxTextRowHeight](#), [Row](#), [TypeEditMultiLine](#), [TypeTextWordWrap](#) properties

## MaxTextCellWidth Property

[See Also](#)

[Example](#)

### Description

Returns the minimum width of the specified cell necessary to display all the text within the cell. This property is available at run time only.

### Syntax

C++ (OCX)            **double** CSpreadSheet::GetMaxTextCellWidth( );

C++ (VBX)            **short** CSpreadSheet::GetMaxTextCellWidth( );

Visual Basic        [form.]Spread.MaxTextCellWidth

### Remarks

Use the [Col](#) and [Row](#) properties to specify the cell.

For edit and static text cells, the value returned varies depending on the setting of the [TypeEditMultiLine](#) property (for edit cells) or the [TypeTextWordWrap](#) property (for static text cells), as listed in the following table.

TypeEditMultiLine or TypeTextWordWrap property setting	Value returned
True	Current cell width
False	The minimum cell width necessary to display the entire line of text with no word wrap given the current font size

### Data Type

VBX: Single

OCX: Double

[Print](#)

[Copy](#)

[Close](#)

The following example determines the maximum width of a cell using the `MaxTextCellWidth` property and assigns this value as the column width using the `ColWidth` property.

C++

```
// Size col width to fill max length of the text in the cell
vaSpread1->SetCol(1);
vaSpread1->SetRow(1);
fWidth = vaSpread1->GetMaxTextCellWidth( );
vaSpread1->SetColWidth(1, fWidth);
```

Visual Basic

```
' Size col width to fill max length of the text in the cell
vaSpread1.Col = 1
vaSpread1.Row = 1
fWidth = vaSpread1.MaxTextCellWidth
vaSpread1.ColWidth(1) = fWidth
```

**See Also**

[AutoSize](#), [Col](#), [MaxTextCellHeight](#), [MaxTextColWidth](#), [MaxTextRowHeight](#), [Row](#), [TypeEditMultiLine](#), [TypeTextWordWrap](#)  
properties



---

## MaxTextColWidth Property

[See Also](#)

[Example](#)

### Description

Returns the width of the widest text string in the specified column. This property is available at run time only.

### Syntax

C++ (OCX)      **double** CSpreadSheet::GetMaxTextColWidth(**long** *Index*);

C++ (VBX)      **short** CSpreadSheet::GetMaxTextColWidth(**short** *Index*);

Visual Basic    [*form.*]Spread.MaxTextColWidth(*Index*)

### Remarks

Specify the column using the *Index* parameter.

Columns with cells that display multiple lines of text return only the current column width. That is, the column width is returned, not the width of the widest text string.

### Data Type

VBX: Single Array

OCX: Double

[Print](#)

[Copy](#)

[Close](#)

The following example determines the maximum width of the second spreadsheet column using the `MaxTextColWidth` property and assigns this value as the column width using the `ColWidth` property.

**C++**

```
// Size column width to fill max length of the text in the
// column
fWidth = vaSpread1->GetMaxTextColWidth(2);
vaSpread1->SetColWidth(2, fWidth);
```

**Visual Basic**

```
' Size column width to fill max length of the text in the
' column
fWidth = vaSpread1.MaxTextColWidth(2)
vaSpread1.ColWidth(2) = fWidth
```

**See Also**

[DAutoSizeCols](#), [MaxTextCellHeight](#), [MaxTextCellWidth](#), [MaxTextRowHeight](#) properties

---

## MaxTextRowHeight Property

[See Also](#)

[Example](#)

### Description

Returns the height of the tallest text string in the specified row. This property is available at run time only.

### Syntax

C++ (OCX)      **double** CSpreadSheet::GetMaxTextRowHeight(**long** *Index*);

C++ (VBX)      **short** CSpreadSheet::GetMaxTextRowHeight(**short** *Index*);

Visual Basic      [*form.*]Spread.MaxTextRowHeight(*Index*)

### Remarks

Specify the row using the *Index* parameter.

### Data Type

VBX: Single Array

OCX: Double

[Print](#)

[Copy](#)

[Close](#)

The following example determines the maximum height of the first spreadsheet row using the `MaxTextRowHeight` property and assigns this value as the row height using the `RowHeight` property.

**C++**

```
// Size row height to fill max height of the text in the row
fHeight = vaSpread1->GetMaxTextRowHeight(1);
vaSpread1->SetRowHeight(1, fHeight);
```

**Visual Basic**

```
' Size row height to fill max height of the text in the row
fHeight = vaSpread1.MaxTextRowHeight(1)
vaSpread1.RowHeight(1) = fHeight
```

**See Also**

[MaxTextCellHeight](#), [MaxTextCellWidth](#), [MaxTextColWidth](#) properties

---

## MoveActiveOnFocus Property

[Example](#)

### Description

Sets or returns the location of the active cell when the user moves the focus to the spreadsheet with the mouse.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetMoveActiveOnFocus( ); <b>void</b> CSpreadSheet::SetMoveActiveOnFocus( <b>BOOL</b> value);
Visual Basic	[form.]Spread.MoveActiveOnFocus[ = boolean%]

### OCX Use

On Focus Set Cell to Cursor check box on the [General](#) property page

### Remarks

The default value for the MoveActiveOnFocus property is True.

When the MoveActiveOnFocus property is set to True, the active cell is moved to the location of the pointer when the user moves the focus to the spreadsheet with the mouse. When the MoveActiveOnFocus property is set to False, the active cell remains where it was before the user moved the focus.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example specifies that when the user moves the focus to the spreadsheet, the cell the user clicks becomes the active cell.

**C++**

```
// Set cell to become active when clicked when the
// spreadsheet receives focus
vaSpread1->SetMoveActiveOnFocus(TRUE);
```

**Visual Basic**

```
' Set cell to become active when clicked when the
' spreadsheet receives focus
vaSpread1.MoveActiveOnFocus = True
```



---

## MultiSelCount Property

[See Also](#)

[Example](#)

### Description

Returns the number of distinct blocks currently selected when the spreadsheet is in normal operation mode and the [AllowMultiBlocks](#) property is set to True. This property is available at run time only.

### Syntax

C++                    **long** CSpreadSheet::GetMultiSelCount( );

Visual Basic        [*form.*]Spread.MultiSelCount

### Data Type

Long Integer

**See Also**

[Action](#) (17 - Add MultiSel Blocks, 18 - Get MultiSel Blocks), [AllowMultiBlocks](#), [MultiSelIndex](#) properties

---

## MultiSelIndex Property

[See Also](#)

[Example](#)

### Description

Sets or returns the coordinates of a block of cells selected when the spreadsheet is in normal operation mode and the [AllowMultiBlocks](#) property is set to True. This property is available at run time only.

### Syntax

C++ **long** CSpreadSheet::GetMultiSelIndex( );  
**void** CSpreadSheet::SetMultiSelIndex(**long** value);

Visual Basic [form.]Spread.MultiSelIndex[ = value&]

### Remarks

When the application performs an action specified by the [Action](#) property, the [MultiSelCount](#) property contains the number of currently selected blocks of cells. The MultiSelIndex property is then used to retrieve each block.

This property is zero based.

Set the MultiSelIndex property to a value less than the MultiSelCount property value, then read the [Col](#), [Row](#), [Col2](#), and [Row2](#) properties to determine the coordinates of the block.

**Note** The MultiSelIndex property returns the coordinates for the block of cells that was *last* selected by the Action property. For example, suppose you select the first three columns using the Action property and then deselect the third column using the mouse. If you use the MultiSelIndex property to retrieve the coordinates, you will get the coordinates for the first three columns (the original block that was defined).

### Data Type

Long Integer

**See Also**

[Action](#) (17 - Add MultiSel Blocks, 18 - Get MultiSel Blocks), [Col](#), [Col2](#), [MultiSelCount](#), [Row](#), [Row2](#) properties

---

## Name Property

### Description

Sets or returns the name used in code to identify the control. This property can only be set at design time and can only be read at run time.

### Syntax

C++ (MFC VBX)	<b>CString</b> CSpreadSheet::GetName( ); <b>void</b> CSpreadSheet::SetName( <b>LPCSTR</b> value);
C++ (OWL VBX)	<b>string</b> CSpreadSheet::GetName( ); <b>void</b> CSpreadSheet::SetName( <b>LPCSTR</b> value);
Visual Basic	[form.]Spread.Name

### OCX Use

Use the container's browser.

### Remarks

The default name for a new control is the control name followed by a unique integer, for example, Spread1.

**Note** Changing a control's name can affect existing code that uses the name.

A control's name must start with a letter and can contain a maximum of 40 characters. It can include numbers and underscore characters, but it cannot include punctuation or spaces.

In Visual Basic, a name can be the same as a reserved word, a property name, or the name of another object, but this can create conflicts in your code.

Different controls cannot share the same name.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

String

---

## NoBeep Property

[See Also](#)   [Example](#)

### Description

Sets or returns whether the spreadsheet sounds warning beeps.

### Syntax

C++                    **BOOL** CSpreadSheet::GetNoBeep( );  
                         **void** CSpreadSheet::SetNoBeep(**BOOL** value);

Visual Basic        [form.]Spread.NoBeep[ = *boolean%*]

### OCX Use

Turn Beep Off check box on the [Op Mode](#) property page

### Remarks

The default value for the NoBeep property is False.

By default, the spreadsheet beeps to warn the user of an error. When the NoBeep property is set to True, the spreadsheet does not beep.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example turns off beeping.

C++

```
// Turn beeping off  
vaSpread1->SetNoBeep(TRUE);
```

Visual Basic

```
' Turn beeping off  
vaSpread1.NoBeep = True
```

**See Also**

[RestrictCols](#), [RestrictRows](#) properties



---

## NoBorder Property

[See Also](#)    [Example](#)

### Description

Sets or returns whether the spreadsheet displays a border on the bottom and right edges.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetNoBorder( ); <b>void</b> CSpreadSheet::SetNoBorder( <b>BOOL</b> value);
Visual Basic	[form.]Spread.NoBorder[ = boolean%]

### OCX Use

Hide Border check box on the [Op Mode](#) property page

### Remarks

The default value for the NoBorder property is False, which displays a border on the bottom and right edges of the spreadsheet.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example hides the bottom and right spreadsheet borders.

C++

```
// Hide the bottom and right borders of the spreadsheet  
vaSpread1->SetNoBorder(TRUE);
```

Visual Basic

```
' Hide the bottom and right borders of the spreadsheet  
vaSpread1.NoBorder = True
```

**See Also**

[PrintBorder](#) property

## OperationMode Property

[See Also](#)

[Example](#)

### Description

Sets or returns the operation mode of the spreadsheet.

### Syntax

C++ (OCX)      **long** CSpreadSheet::GetOperationMode( );  
                  **void** CSpreadSheet::SetOperationMode(**long** value);

C++ (VBX)      **short** CSpreadSheet::GetOperationMode( );  
                  **void** CSpreadSheet::SetOperationMode(**short** value);

Visual Basic    [form.]Spread.OperationMode[ = setting%]

### OCX Use

Normal, Read Only, Row Mode, Single Selection, Multiple Selection, or Extended Selection option buttons under Operation Mode on the [Op.Mode](#) property page

### Remarks

The following settings are available:

Setting	Description	Constant
0 - Normal	(Default) Performs default spreadsheet operations	SS_OP_MODE_NORMAL
1 - Read Only	Places spreadsheet in read-only mode (Within this mode, there is no current active cell.)	SS_OP_MODE_READONLY
2 - Row Mode	Places spreadsheet in row mode. A bar appears on the current active row (When the user begins typing data, the data is entered in the current row. Once the user leaves the row, the highlight bar reappears. The <a href="#">EnterRow</a> and <a href="#">LeaveRow</a> events both occur when this mode is set.)	SS_OP_MODE_ROWMODE
3 - Single Select	Sets spreadsheet to operate like a single-selection list box	SS_OP_MODE_SINGLE_SELECT
4 - Multi Select	Sets spreadsheet to operate like a multiple-selection list box	SS_OP_MODE_MULTI_SELECT
5 - Extended Select	Sets spreadsheet to operate like an extended-selection list box	SS_OP_MODE_EXT_SELECT

### Data Type

Integer (Enumerated)

[Print](#)

[Copy](#)

[Close](#)

The following example sets the spreadsheet's operation mode to single selection.

C++

```
// Set spreadsheet's operation mode  
vaSpread1->SetOperationMode(SS_OP_MODE_SINGLE_SELECT);
```

Visual Basic

```
' Set spreadsheet's operation mode  
vaSpread1.OperationMode = SS_OP_MODE_SINGLE_SELECT
```

**See Also**

[MultiSelCount](#), [MultiSelIndex](#), [SelModeIndex](#), [SelModeSelCount](#), [SelModeSelected](#) properties

[EnterRow](#), [LeaveRow](#) events

---

## Parent Property

### Description

Returns the object on which a control is located. This property is available at run time only.

### Syntax

Visual Basic      *[form.]Spread.Parent*

### OCX Use

The Parent property is a standard extender property that can be provided by your container.

### Remarks

Use the Parent property to access the properties, methods, or controls of a control's parent object.

In C++, the parent should be retrieved using Windows API calls. MFC users can use the `CWnd::GetParent` function. OWL users can use the `TWindow::GetParent` function.

The Parent property is useful in an application in which you pass controls as arguments. For example, in Visual Basic, you might pass a control variable to a general procedure in a module, and use the Parent property to access its parent object.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

Object

## Position Property

[See Also](#)   [Example](#)

### Description

Sets or returns the desired position of a cell on the screen.

### Syntax

C++                    **short** CSpreadSheet::GetPosition( );  
                         **void** CSpreadSheet::SetPosition(**short** value);

Visual Basic        [form.]Spread.Position[ = setting%]

### Remarks

Use the Position property with the [Action](#), [Col](#), and [Row](#) properties to specify the location of a cell on the screen.

The following settings are available:

Setting	Constant
0 - Upper Left (Default)	SS_POSITION_UPPER_LEFT
1 - Upper Center	SS_POSITION_UPPER_CENTER
2 - Upper Right	SS_POSITION_UPPER_RIGHT
3 - Center Left	SS_POSITION_CENTER_LEFT
4 - Center	SS_POSITION_CENTER_CENTER
5 - Center Right	SS_POSITION_CENTER_RIGHT
6 - Bottom Left	SS_POSITION_BOTTOM_LEFT
7 - Bottom Center	SS_POSITION_BOTTOM_CENTER
8 - Bottom Right	SS_POSITION_BOTTOM_RIGHT

### Data Type

Integer (Enumerated)



[Print](#)

[Copy](#)

[Close](#)

The following example moves a cell to the upper-left corner of the spreadsheet.

C++

```
// Scroll the spreadsheet to the desired cell
vaSpread1->SetRow(20);
vaSpread1->SetCol(20);
vaSpread1->SetPosition(SS_POSITION_UPPER_LEFT);
vaSpread1->SetAction(SS_ACTION_GOTO_CELL);
```

Visual Basic

```
' Scroll the spreadsheet to the desired cell
vaSpread1.Row = 20
vaSpread1.Col = 20
vaSpread1.Position = SS_POSITION_UPPER_LEFT
vaSpread1.Action = SS_ACTION_GOTO_CELL
```

**See Also**

[Action](#) (1 - Go To), [Col](#), [Row](#) properties

---

## PrintAbortMsg Property

[See Also](#)

[Example](#)

### Description

Sets or returns the information displayed in an abort dialog box during printing. This property is available at run time only.

### Syntax

C++ (OCX)	<b>CString</b> CSpreadSheet::GetPrintAbortMsg( ); <b>void</b> CSpreadSheet::SetPrintAbortMsg( <b>LPCSTR</b> value);
C++ (MFC VBX)	<b>CString</b> CSpreadSheet::GetPrintAbortMsg( ); <b>void</b> CSpreadSheet::SetPrintAbortMsg( <b>LPCSTR</b> value);
C++ (OWL VBX)	<b>string</b> CSpreadSheet::GetPrintAbortMsg( ); <b>void</b> CSpreadSheet::SetPrintAbortMsg( <b>LPCSTR</b> value);
Visual Basic	[form.]Spread.PrintAbortMsg[ = text\$]

### Remarks

The PrintAbortMsg property determines the message string used in an abort dialog box displayed during printing that lets the user cancel printing. If no string is supplied for the PrintAbortMsg property, the abort dialog box is not displayed.

### Data Type

String

[Print](#)

[Copy](#)

[Close](#)

The following example sets the message string to be used in a print abort dialog box.

C++

```
// Define the text for the print abort message  
vaSpread1->SetPrintAbortMsg("This is the PRINT abort message.");
```

Visual Basic

```
' Define the text for the print abort message  
vaSpread1.PrintAbortMsg = "This is the PRINT abort message."
```

**See Also**

[Action](#) (13 - Print, 32 - Smart Print) property

---

## PrintBorder Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether to print the spreadsheet border. This property is available at run time only.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetPrintBorder( ); <b>void</b> CSpreadSheet::SetPrintBorder( <b>BOOL</b> value);
Visual Basic	[form.]Spread.PrintBorder[ = boolean%]

### Remarks

The default value for the PrintBorder property is True, which prints a border on the spreadsheet.

Use the PrintBorder property with the [Action](#) property.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example specifies that the spreadsheet border, row and column headers, and grid lines are not printed.

C++

```
// Do not print the border, headers, or grid lines
vaSpread1->SetPrintBorder(FALSE);
vaSpread1->SetPrintColHeaders(FALSE);
vaSpread1->SetPrintGrid(FALSE);
vaSpread1->SetPrintRowHeaders(FALSE);
```

Visual Basic

```
' Do not print the border, headers, or grid lines
vaSpread1.PrintBorder = False
vaSpread1.PrintColHeaders = False
vaSpread1.PrintRowHeaders = False
vaSpread1.PrintGrid = False
```

**See Also**

[Action](#) (13 - Print, 32 - Smart Print), [NoBorder](#) properties



---

## PrintColHeaders Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether to print the column header row at the top of each page. This property is available at run time only.

### Syntax

C++ **BOOL** CSpreadSheet::GetPrintColHeaders( );  
**void** CSpreadSheet::SetPrintColHeaders(**BOOL** value);

Visual Basic [form.]Spread.PrintColHeaders[ = boolean%]

### Remarks

The default value for the PrintColHeaders property is True, which prints the spreadsheet's column header row.

Use the PrintColHeaders property with the [Action](#) property.

### Data Type

Integer (Boolean)

**See Also**

[Action](#) (13 - Print, 32 - Smart Print) property

---

## PrintColor Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether to print the colors in the spreadsheet as they appear on the screen. This property is available at run time only.

### Syntax

C++ **BOOL** CSpreadSheet::GetPrintColor( );  
**void** CSpreadSheet::SetPrintColor(**BOOL** value);

Visual Basic [form.]Spread.PrintColor[ = boolean%]

### Remarks

The default value for the PrintColor property is False.

When the PrintColor property is set to False, all colors specified within the spreadsheet are ignored. When the PrintColor property is set to True and the spreadsheet is not printed on a color printer, the colors print as dithered patterns.

Use the PrintColor property with the [Action](#) property.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example specifies that the colors in the spreadsheet are printed as they appear on the screen.

C++

```
// Set to print on a color printer  
vaSpread1->SetPrintColor(TRUE);
```

Visual Basic

```
' Set to print on a color printer  
vaSpread1.PrintColor = True
```

**See Also**

[Action](#) (13 - Print, 32 - Smart Print) property

## PrintFooter Property

[See Also](#)

[Example](#)

### Description

Sets or returns the text and format of footers on printed pages. This property is available at run time only.

### Syntax

C++ (OCX)      **CString** CSpreadSheet::GetPrintFooter( );  
                  **void** CSpreadSheet::SetPrintFooter(**LPCWSTR** value);

C++ (MFC VBX)    **CString** CSpreadSheet::GetPrintFooter( );  
                  **void** CSpreadSheet::SetPrintFooter(**LPCWSTR** value);

C++ (OWL VBX)    **string** CSpreadSheet::GetPrintFooter( );  
                  **void** CSpreadSheet::SetPrintFooter(**LPCWSTR** value);

Visual Basic      [form.]Spread.PrintFooter[ = text\$]

### Remarks

Use the PrintFooter property with the [Action](#) property.

The following control characters are available:

Escape Code	Description
/n	New line
/l	Left aligns text
/r	Right aligns text
/c	Centers text
/p	Inserts page number
//	Prints a slash character (/)
/fn	Font name, must be enclosed in quotes
/fz	Font size, must be enclosed in quotes
/fb0	Font bold off
/fb1	Font bold on
/fi0	Font italics off
/fi1	Font italics on
/fu0	Font underline off
/fu1	Font underline on
/fk0	Font strikethrough off
/fk1	Font strikethrough on
/fs1	Saves the current font configuration as configuration 1
/fs2	Saves the current font configuration as configuration 2
/fs3	Saves the current font configuration as configuration 3
/fs4	Saves the current font configuration as configuration 4
/fs5	Saves the current font configuration as configuration 5
/fs6	Saves the current font configuration as configuration 6
/fs7	Saves the current font configuration as configuration 7
/fs8	Saves the current font configuration as configuration 8
/fs9	Saves the current font configuration as configuration 9
/f0	Resets the font to the spreadsheet default font
/f1	Recalls font configuration 1
/f2	Recalls font configuration 2
/f3	Recalls font configuration 3
/f4	Recalls font configuration 4
/f5	Recalls font configuration 5
/f6	Recalls font configuration 6
/f7	Recalls font configuration 7
/f8	Recalls font configuration 8
/f9	Recalls font configuration 9

### Data Type

String



[Print](#)

[Copy](#)

[Close](#)

The following example defines two font configurations and uses them to set the spreadsheet header and footer text.

C++

```
// Create a font as Arial, size 10, bold, no italics,  
// underline, no strikethrough and save it as font #1  
lstrcpy(font1, "/fn\"Arial\"/fz\"10\" /fb1 /fi0 /fu1 /fk0 /fs1");  
// Create a font as Times, size 20, no bold, italics, no  
// underline, strikethrough and save it as font #2  
lstrcpy(font2, "/fn\"Times\"/fz\"20\" /fb0 /fi1 /fu0 /fk1 /fs2");  
// Recall font configurations and set the header and footer text  
wsprintf(buf, "%s%s/f1This is font #1 \n/f2this is font #2", font1, font2);  
vaSpread1->SetPrintFooter(buf);  
vaSpread1->SetPrintHeader(buf);
```

Visual Basic

```
' Create a font as Arial, size 10, bold, no italics,  
' underline, no strikethrough and save it as font #1  
font1 = "/fn\"Arial\" /fz\"10\"/fb1 /fi0 /fu1 /fk0 /fs1"  
' Create a font as Times, size 20, no bold, italics, no  
' underline, strikethrough and save it as font #2  
font2 = "/fn\"Times\" /fz\"20\"/fb0 /fi1 /fu0 /fk1 /fs2"  
' Recall font configurations and set the header and footer text  
buf = font1 + font2 + "/f1This is font #1 "+Chr$(13) + "/f2This is font #2"  
vaSpread1.PrintFooter = buf  
vaSpread1.PrintHeader = buf
```



**See Also**

[Action](#) (13 - Print, 32 - Smart Print) property

---

## PrintGrid Property

[See Also](#)    [Example](#)

### Description

Sets or returns whether to print the spreadsheet grid lines. This property is available at run time only.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetPrintGrid( ); <b>void</b> CSpreadSheet::SetPrintGrid( <b>BOOL</b> value);
Visual Basic	[form.]Spread.PrintGrid[ = boolean%]

### Remarks

The default value for the PrintGrid property is True, which prints the grid lines the way the spreadsheet displays them.

Use the PrintGrid property with the [Action](#) property.

### Data Type

Integer (Boolean)

**See Also**

[Action](#) (13 - Print, 32 - Smart Print) property

## PrintHeader Property

[See Also](#)

[Example](#)

### Description

Sets or returns the text and format of headers on printed pages. This property is available at run time only.

### Syntax

C++ (OCX)      **CString** CSpreadSheet::GetPrintHeader( );  
                 **void** CSpreadSheet::SetPrintHeader(**LPCWSTR** value);

C++ (MFC VBX)      **CString** CSpreadSheet::GetPrintHeader( );  
                 **void** CSpreadSheet::SetPrintHeader(**LPCSTR** value);

C++ (OWL VBX)      **string** CSpreadSheet::GetPrintHeader( );  
                 **void** CSpreadSheet::SetPrintHeader(**LPCSTR** value);

Visual Basic      [form.]Spread.PrintHeader[ = text\$]

### Remarks

Use the PrintHeader property with the [Action](#) property.

The following control characters are available:

Escape Code	Description
/n	New line
/l	Left aligns text
/r	Right aligns text
/c	Centers text
/p	Inserts page number
//	Prints a slash character (/)
/fn	Font name, must be enclosed in quotes
/fz	Font size, must be enclosed in quotes
/fb0	Font bold off
/fb1	Font bold on
/fi0	Font italics off
/fi1	Font italics on
/fu0	Font underline off
/fu1	Font underline on
/fk0	Font strikethrough off
/fk1	Font strikethrough on
/fs1	Saves the current font configuration as configuration 1
/fs2	Saves the current font configuration as configuration 2
/fs3	Saves the current font configuration as configuration 3
/fs4	Saves the current font configuration as configuration 4
/fs5	Saves the current font configuration as configuration 5
/fs6	Saves the current font configuration as configuration 6
/fs7	Saves the current font configuration as configuration 7
/fs8	Saves the current font configuration as configuration 8
/fs9	Saves the current font configuration as configuration 9
/f0	Resets the font to the spreadsheet default font
/f1	Recalls font configuration 1
/f2	Recalls font configuration 2
/f3	Recalls font configuration 3
/f4	Recalls font configuration 4
/f5	Recalls font configuration 5
/f6	Recalls font configuration 6
/f7	Recalls font configuration 7
/f8	Recalls font configuration 8
/f9	Recalls font configuration 9

### Data Type

String



**See Also**

[Action](#) (13 - Print, 32 - Smart Print) property

---

## PrintJobName Property

[See Also](#)

[Example](#)

### Description

Sets or returns the print job name displayed in the Print Manager when printing the spreadsheet. This property is available at run time only.

### Syntax

C++ (OCX)	<b>CString</b> CSpreadSheet::GetPrintJobName( ); <b>void</b> CSpreadSheet::SetPrintJobName( <b>LPCWSTR</b> value);
C++ (MFC VBX)	<b>CString</b> CSpreadSheet::GetPrintJobName( ); <b>void</b> CSpreadSheet::SetPrintJobName( <b>LPCSTR</b> value);
C++ (OWL VBX)	<b>string</b> CSpreadSheet::GetPrintJobName( ); <b>void</b> CSpreadSheet::SetPrintJobName( <b>LPCSTR</b> value);
Visual Basic	[form.]Spread.PrintJobName[ = text\$]

### Remarks

The PrintJobName property lets the user identify the spreadsheet being printed.

The default text for the PrintJobName property is "Spread".

Use the PrintJobName property with the [Action](#) property.

### Data Type

String

[Print](#)

[Copy](#)

[Close](#)

The following example sets the print job name.

C++

```
// Define the print job name  
vaSpread1->SetPrintJobName("FarPoint print job");
```

Visual Basic

```
' Define the print job name  
vaSpread1.PrintJobName = "FarPoint print job"
```



**See Also**

[Action](#) (13 - Print, 32 - Smart Print) property

---

## PrintMarginBottom Property

[See Also](#)    [Example](#)

### Description

Sets or returns the bottom margin when printing the spreadsheet. This property is available at run time only.

### Syntax

C++	<b>long</b> CSpreadSheet::GetPrintMarginBottom( ); <b>void</b> CSpreadSheet::SetPrintMarginBottom( <b>long</b> value);
Visual Basic	[form.]Spread.PrintMarginBottom[ = value&]

### Remarks

The PrintMarginBottom property lets the application determine the bottom margin to use when printing a page. The units are specified in twips.

The default value for the PrintMarginBottom property is 0.

Use the PrintMarginBottom property with the [Action](#) property.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets the spreadsheet page margins to 1440 twips (1 inch).

C++

```
// Define the page margins as 1 inch
vaSpread1->SetPrintMarginBottom(1440);
vaSpread1->SetPrintMarginLeft(1440);
vaSpread1->SetPrintMarginRight(1440);
vaSpread1->SetPrintMarginTop(1440);
```

Visual Basic

```
' Define the page margins as 1 inch
vaSpread1.PrintMarginBottom = 1440
vaSpread1.PrintMarginLeft = 1440
vaSpread1.PrintMarginRight = 1440
vaSpread1.PrintMarginTop = 1440
```

**See Also**

[Action](#) (13 - Print, 32 - Smart Print) property

---

## PrintMarginLeft Property

[See Also](#)   [Example](#)

### Description

Sets or returns the left margin when printing the spreadsheet. This property is available at run time only.

### Syntax

C++	<b>long</b> CSpreadSheet::GetPrintMarginLeft( ); <b>void</b> CSpreadSheet::SetPrintMarginLeft( <b>long</b> value);
Visual Basic	[form.]Spread.PrintMarginLeft[ = value&]

### Remarks

The PrintMarginLeft property lets the application determine the left margin to use when printing a page. The units are specified in twips.

The default value for the PrintMarginLeft property is 0.

Use the PrintMarginLeft property with the [Action](#) property.

### Data Type

Long Integer

**See Also**

[Action](#) (13 - Print, 32 - Smart Print) property

---

## PrintMarginRight Property

[See Also](#)   [Example](#)

### Description

Sets or returns the right margin when printing the spreadsheet. This property is available at run time only.

### Syntax

C++	<b>long</b> CSpreadSheet::GetPrintMarginRight( ); <b>void</b> CSpreadSheet::SetPrintMarginRight( <b>long</b> value);
Visual Basic	[form.]Spread.PrintMarginRight[ = value&]

### Remarks

The PrintMarginRight property lets the application determine the right margin to use when printing a page. The units are specified in twips.

The default value for the PrintMarginRight property is 0.

Use the PrintMarginRight property with the [Action](#) property.

### Data Type

Long Integer

**See Also**

[Action](#) (13 - Print, 32 - Smart Print) property



---

## PrintMarginTop Property

[See Also](#)

[Example](#)

### Description

Sets or returns the top margin when printing the spreadsheet. This property is available at run time only.

### Syntax

C++	<b>long</b> CSpreadSheet::GetPrintMarginTop( ); <b>void</b> CSpreadSheet::SetPrintMarginTop( <b>long value</b> );
Visual Basic	[ <i>form.</i> ]Spread.PrintMarginTop[ = <i>value</i> &]

### Remarks

The PrintMarginTop property lets the application determine the top margin to use when printing a page. The units are specified in twips.

The default value for the PrintMarginTop property is 0.

Use the PrintMarginTop property with the [Action](#) property.

### Data Type

Long Integer

**See Also**

[Action](#) (13 - Print, 32 - Smart Print) property

## PrintOrientation Property

[See Also](#)

### Description

Sets or returns the page orientation used to print the spreadsheet. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetPrintOrientation( );  
                         **void** CSpreadSheet::SetPrintOrientation(**short** value);

Visual Basic        [form.]Spread.PrintOrientation[ = setting%]

### Remarks

The following settings are available:

Setting	Description	Constant
0 - Default	(Default) Uses the current printer setting	SS_PRINTORIENT_DEFAULT
1 - Portrait	Prints spreadsheet in portrait mode	SS_PRINTORIENT_PORTRAIT
2 - Landscape	Prints spreadsheet in landscape mode	SS_PRINTORIENT_LANDSCAPE

Use the PrintOrientation property with the [Action](#) property.

### Data Type

Integer (Enumerated)

**See Also**

[Action](#) (13 - Print, 32 - Smart Print) property

---

## PrintPageEnd, PrintPageStart Properties

[See Also](#)    [Example](#)

### Description

Set or return the first and last pages of the spreadsheet to be printed. These properties are available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetPrintPageEnd( );  
                         **void** CSpreadSheet::SetPrintPageEnd(**short** value);

Visual Basic        [form.]Spread.PrintPageEnd[ = value%]

**Note**    The PrintPageStart property uses similar syntax.

### Remarks

The PrintPageStart and PrintPageEnd properties let the application specify a range of pages to be printed. The application must set the [PrintType](#) property to 3 (Page Range).

Use the PrintPageEnd and PrintPageStart properties with the [Action](#) property.

### Data Type

Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets spreadsheet page 3 as the first page to print and page 10 as the last page to print.

C++

```
// Define the last page to print  
vaSpread1->SetPrintPageStart(3);  
vaSpread1->SetPrintPageEnd(10);
```

Visual Basic

```
' Define the last page to print  
vaSpread1.PrintPageStart = 3  
vaSpread1.PrintPageEnd = 10
```

**See Also**

[Action](#) (13 - Print, 32 - Smart Print), PrintType properties

---

## PrintRowHeaders Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether to print the row header column on the left side of each page. This property is available at run time only.

### Syntax

C++                    **BOOL** CSpreadSheet::GetPrintRowHeaders( );  
                         **void** CSpreadSheet::SetPrintRowHeaders(**BOOL** value);

Visual Basic        [form.]Spread.PrintRowHeaders[ = *boolean%*]

### Remarks

The default value for the PrintRowHeaders property is True.

When the PrintRowHeaders property is set to True, the row header column is printed with the spreadsheet.

Use the PrintRowHeaders property with the [Action](#) property.

### Data Type

Integer (Boolean)



**See Also**

[Action](#) (13 - Print, 32 - Smart Print) property

---

## PrintShadows Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether to print the shadow effect within the spreadsheet row and column headers. This property is available at run time only.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetPrintShadows( ); <b>void</b> CSpreadSheet::SetPrintShadows( <b>BOOL</b> value);
Visual Basic	[form.]Spread.PrintShadows[ = boolean%]

### Remarks

The default value for the PrintShadows property is True.

When the PrintShadows property is set to True, the shadow effect within the row and column headers is printed when the spreadsheet is printed. Setting the PrintShadows property to False makes the spreadsheet print more rapidly.

Use the PrintShadows property with the [Action](#) property.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example specifies that no shadow effect is printed within the spreadsheet row and column headers.

C++

```
// Do not print the shadow effect  
vaSpread1->SetPrintShadows (FALSE);
```

Visual Basic

```
' Do not print the shadow effect  
vaSpread1.PrintShadows = False
```

**See Also**

[Action](#) (13 - Print, 32 - Smart Print) property

## PrintType Property

[See Also](#)    [Example](#)

### Description

Sets or returns the type of print range used to print the spreadsheet. This property is available at run time only.

### Syntax

```
C++          short CSpreadSheet::GetPrintType( );  
            void CSpreadSheet::SetPrintType(short value);  
  
Visual Basic  [form.]Spread.PrintType[ = setting%]
```

### Remarks

You can use a variety of options to determine the range type for printing the spreadsheet. Depending upon the range type you specify with the PrintType property, other properties can be used to specify the range. The PrintType, [PrintPageStart](#), [PrintPageEnd](#), [Col](#), [Row](#), [Col2](#), and [Row2](#) properties define the area of the spreadsheet to print. However, if the [PrintUseDataMax](#) property is set to True, data beyond the area defined by the [DataColCnt](#) and [DataRowCnt](#) properties will not be printed.

The following settings are available:

Setting	Description	Constant
0 - All	(Default) Prints the entire spreadsheet	SS_PRINT_ALL
1 - Cell Range	Prints the area specified by the Col, Col2, Row, and Row2 properties	SS_PRINT_CELL_RANGE
2 - Current Page	Prints the current page of the spreadsheet	SS_PRINT_CURRENT_PAGE
3 - Page Range	Prints the pages in the range specified by the PrintPageStart and PrintPageEnd properties	SS_PRINT_PAGE_RANGE

Use the PrintType property with the [Action](#) property.

### Data Type

Integer (Enumerated)

[Print](#)

[Copy](#)

[Close](#)

The following example specifies a cell range as the print range.

**C++**

```
// Select a block of cells
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
vaSpread1->SetRow2(4);
vaSpread1->SetCol2(4);
// Set the cell range to be printed
vaSpread1->SetPrintType(SS_PRINT_CELL_RANGE);
```

**Visual Basic**

```
' Select a block of cells
vaSpread1.Row = 2
vaSpread1.Col = 2
vaSpread1.Row2 = 4
vaSpread1.Col2 = 4
' Set the cell range to be printed
vaSpread1.PrintType = SS_PRINT_CELL_RANGE
```

**See Also**

[Action](#) (13 - Print, 32 - Smart Print), [Col](#), [Col2](#), [DataColCnt](#), [DataRowCnt](#), [PrintPageStart](#), [PrintPageEnd](#), [PrintUseDataMax](#), [Row](#), [Row2](#) properties

---

## PrintUseDataMax Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the number of rows and columns that are printed is no greater than the number of rows and columns containing data. This property is available at run time only.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetPrintUseDataMax( ); <b>void</b> CSpreadSheet::SetPrintUseDataMax( <b>BOOL</b> value);
Visual Basic	[form.]Spread.PrintUseDataMax[ = boolean%]

### Remarks

The default value for the PrintUseDataMax property is True.

When the PrintUseDataMax property is set to True, the number of rows and columns that print is not greater than the number of rows and columns containing data. When the PrintUseDataMax property is set to False, the number of rows and columns that print is defined by the [PrintType](#), [PrintPageStart](#), [PrintPageEnd](#), [Col](#), [Row](#), [Col2](#), and [Row2](#) properties.

Use the PrintUseDataMax property with the [Action](#) property.

### Data Type

Integer (Boolean)



[Print](#)

[Copy](#)

[Close](#)

The following example specifies that only the rows and columns containing data are printed.

C++

```
// Print the portion of the spreadsheet that contains data  
vaSpread1->SetPrintUseDataMax(TRUE);
```

Visual Basic

```
' Print the portion of the spreadsheet that contains data  
vaSpread1.PrintUseDataMax = True
```

**See Also**

[Action](#) (13 - Print, 32 - Smart Print), [Col](#), [Col2](#), [PrintPageEnd](#), [PrintPageStart](#), [PrintType](#), [Row](#), [Row2](#) properties

---

## ProcessTab Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the Tab key moves the focus either to the next cell in the spreadsheet or to the next control on the dialog.

### Syntax

```
C++          BOOL CSpreadSheet::GetProcessTab( );  
            void CSpreadSheet::SetProcessTab(BOOL value);  
Visual Basic  [form.]Spread.ProcessTab[ = boolean%]
```

### OCX Use

Process Tab Key check box on the [Edit Mode](#) property page

### Remarks

The default value for the ProcessTab property is False.

When the ProcessTab property is set to False, the Tab key moves the focus to the next control on the dialog.

When the ProcessTab property is set to True, the user can use the Tab key to move the active cell to the next cell on the right. The active cell is moved from left to right, until the far right cell is reached, then the next cell is the first column of the next row. If Shift+Tab is pressed, the active cell moves from right to left, until the far left cell is reached, and the next cell is the last column of the previous row.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example allows the user to use the Tab key in the spreadsheet to move from cell to cell.

C++

```
// Tab key moves focus to next cell  
vaSpread1->SetProcessTab(TRUE);
```

Visual Basic

```
' Tab key moves focus to next cell  
vaSpread1.ProcessTab = True
```

**See Also**

[EditEnterAction](#) property

[Advance](#) event

---

## Protect Property

[See Also](#)   [Example](#)

### Description

Sets or returns whether the user can edit cells marked as locked with the [Lock](#) property.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetProtect( ); <b>void</b> CSpreadSheet::SetProtect( <b>BOOL</b> value);
Visual Basic	[form.]Spread.Protect[ = boolean%]

### OCX Use

Protect Locked Cells check box on the [Op\\_Mode](#) property page

### Remarks

You must perform three steps to lock cells. First, specify the cells you want to prevent the user from editing. Use the [Col](#) and [Row](#) properties to specify a cell to mark as locked. To mark a block of cells as locked, use the Col, [Col2](#), Row, [Row2](#), and [BlockMode](#) properties to specify the block of cells before setting the [Lock](#) property. Next, mark the cells as locked by setting the Lock property. Complete locking the cells by setting the Protect property to True.

Note that when the Protect property is set to False, the user can still edit cells marked as locked with the Lock property.

The default value for the Protect property is True.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example lets the user edit cells that are marked as locked.

C++

```
// Do not protect cells marked as locked  
vaSpread1->SetProtect (FALSE);
```

Visual Basic

```
' Do not protect cells marked as locked  
vaSpread1.Protect = False
```

**See Also**

[Lock](#) property

[EditError](#) event



---

## ReDraw Property

[Example](#)

### Description

Sets or returns whether the application immediately redraws the Spread control after the user makes changes.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetReDraw( ); <b>void</b> CSpreadSheet::SetReDraw( <b>BOOL</b> value);
Visual Basic	[form.]Spread.ReDraw[ = boolean%]

### OCX Use

Redraw check box on the [Display](#) property page

### Remarks

The default value for the ReDraw property is True.

When the ReDraw property is set to True, the spreadsheet automatically refreshes itself each time the user sets a property that changes the spreadsheet's appearance or data. When the ReDraw property is set to False, the spreadsheet does not refresh itself each time the user makes changes.

Set the ReDraw property to False when adding large amounts of data or when making multiple changes to the spreadsheet appearance; then when you are finished, reset the ReDraw property to True. The items on the screen that have changed since the ReDraw property was set to False are then refreshed.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example sets the ReDraw property to False, places text in three cells, and then sets the ReDraw property back to True to refresh the spreadsheet.

C++

```
vaSpread1->SetReDraw (FALSE) ;  
vaSpread1->SetRow (0) ;  
vaSpread1->SetCol (1) ;  
vaSpread1->SetText ("Home") ;  
vaSpread1->SetCol (2) ;  
vaSpread1->SetText ("Auto") ;  
vaSpread1->SetCol (3) ;  
vaSpread1->SetText ("Phone") ;  
vaSpread1->SetReDraw (TRUE) ;
```

Visual Basic

```
vaSpread1.ReDraw = False  
vaSpread1.Row = 0  
vaSpread1.Col = 1  
vaSpread1.Text = "Home"  
vaSpread1.Col = 2  
vaSpread1.Text = "Auto"  
vaSpread1.Col = 3  
vaSpread1.Text = "Phone"  
vaSpread1.ReDraw = True
```

---

## RestrictCols Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the user can type data in a column that is more than one column beyond the last column containing data.

### Syntax

C++ **BOOL** CSpreadSheet::GetRestrictCols( );  
**void** CSpreadSheet::SetRestrictCols(**BOOL** value);

Visual Basic [form.]Spread.RestrictCols[ = boolean%]

### OCX Use

Restrict Data Entry check box under Col Headers on the [Headers](#) property page

### Remarks

The default value for the RestrictCols property is False, which means that the user can type data in any column in the spreadsheet.

When the RestrictCols property is set to True, the user can type data in all columns up to one column beyond the last column containing data. This ensures that the user types data one column at a time. If the user attempts to type data beyond that point, the spreadsheet beeps (unless the [NoBeep](#) property is set to True).

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example lets the user type data in only one column beyond the last column containing data.

C++

```
// User must type data one column at a time  
vaSpread1->SetRestrictCols(TRUE);
```

Visual Basic

```
' User must type data one column at a time  
vaSpread1.RestrictCols = True
```

**See Also**

[DataColCnt](#), [DataRowCnt](#), [NoBeep](#), [RestrictRows](#) properties

[EditError](#) event

---

## RestrictRows Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the user can type data in a row that is more than one row beyond the last row containing data.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetRestrictRows( ); <b>void</b> CSpreadSheet::SetRestrictRows( <b>BOOL</b> value);
Visual Basic	[form.]Spread.RestrictRows[ = <i>boolean%</i> ]

### OCX Use

Restrict Data Entry check box under Row Headers on the [Headers](#) property page

### Remarks

The default value for the RestrictRows property is False, which means that the user can type data in any row in the spreadsheet.

When the RestrictRows property is set to True, the user can type data in all rows up to one row beyond the last row containing data. This ensures that the user types data one row at a time. If the user attempts to type data beyond that point, the spreadsheet beeps (unless the [NoBeep](#) property is set to True).

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example lets the user type data in only one row beyond the last row containing data.

C++

```
// User must type data one row at a time  
vaSpread1->SetRestrictRows(TRUE);
```

Visual Basic

```
' User must type data one row at a time  
vaSpread1.RestrictRows = True
```

**See Also**

[DataColCnt](#), [DataRowCnt](#), [NoBeep](#), [RestrictCols](#) properties

[EditError](#) event



---

## RetainSelBlock Property

[Example](#)

### Description

Sets or returns whether a selected block of cells in the spreadsheet remains highlighted when the spreadsheet loses the focus.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetRetainSelBlock( ); <b>void</b> CSpreadSheet::SetRetainSelBlock( <b>BOOL</b> value);
Visual Basic	[form.]Spread.RetainSelBlock[ = <i>boolean</i> %]

### OCX Use

Retain Selected Block check box on the [Op.Mode](#) property page

### Remarks

The default value for the RetainSelBlock property is True.

When the RetainSelBlock property is set to True, the selected block of cells retains its highlighting. When the RetainSelBlock property is set to False, the application removes the highlighting from the selected block, but then restores it when the spreadsheet regains the focus. This also applies to the active cell.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example specifies that selected cells do not remain highlighted when the spreadsheet loses the focus.

**C++**

```
// Do not display highlighting on selected cells when  
// spreadsheet loses the focus  
vaSpread1->SetRetainSelBlock(FALSE);
```

**Visual Basic**

```
' Do not display highlighting on selected cells when  
' spreadsheet loses the focus  
vaSpread1.RetainSelBlock = False
```

## Row Property

[See Also](#)   [Example](#)

### Description

Sets or returns a specific row or specifies the first row of a block of cells on which an operation is to occur. This property is available at run time only.

### Syntax

C++                    **long** CSpreadSheet::GetRow( );  
                         **void** CSpreadSheet::SetRow(**long** value);

Visual Basic        [form.]Spread.Row[ = value&]

### Remarks

Use the Row property in conjunction with other properties to specify characteristics for a column or a block of cells. Often, the Row property is used with the [Col](#), [Col2](#), and [Row2](#) properties to specify a block of cells to be affected by block properties, such as [Clip](#) or [BlockMode](#).

To specify the cell or cells to be affected by other properties, first specify the necessary cells using the properties listed in the following table, then set the properties that affect the designated cells.

To specify . . .	Set . . .
A cell	Col, Row
A column	Col to non-zero, Row to —1
A range of columns	Col, Col2
A row	Row to non-zero, Col to —1
A range of rows	Row, Row2
A block of cells	Col, Row, Col2, Row2
The entire spreadsheet	Col to —1 or Row to —1

The default value for the Row property is —1.

The Row property has no association with the current active row on the screen.

The Row property is zero-based. Set the Row property to 0 to specify the column header row; set the Row property to —1 to specify all rows.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets a cell's background color to red.

**C++**

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Define the background color for the cell
// red, RGB(255,0,0)
vaSpread1->SetBackColor(0x000000FF);
```

**Visual Basic**

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Define the background color for the cell
' red, RGB(255,0,0)
vaSpread1.BackColor = &H000000FF&
```

**See Also**

[Action](#), [BlockMode](#), [Clip](#), [Col](#), [Col2](#), [Row2](#) properties

## Row2 Property

[See Also](#)   [Example](#)

### Description

Sets or returns the last row of a block of cells on which an operation is to occur. This property is available at run time only.

### Syntax

C++                    **long** CSpreadSheet::GetRow2( );  
                         **void** CSpreadSheet::SetRow2(**long** value);

Visual Basic        [form.]Spread.Row2[ = value&]

### Remarks

Use the Row2 property in conjunction with other properties to specify characteristics for a column or a block of cells. Often, the Row2 property is used with the [Col](#), [Col2](#), and [Row](#) properties to specify a block of cells to be affected by block properties, such as [Clip](#) or [BlockMode](#).

To specify the cell or cells to be affected by other properties, first specify the necessary cells using the properties listed in the following table, then set the properties that affect the designated cells.

To specify . . .	Set . . .
A cell	Col, Row
A column	Col to non-zero, Row to —1
A range of columns	Col, Col2
A row	Row to non-zero, Col to —1
A range of rows	Row, Row2
A block of cells	Col, Row, Col2, Row2
The entire spreadsheet	Col to —1 or Row to —1

The default value for the Row2 property is 0.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets the background color of a block of cells to red.

C++

```
// Select a block of cells
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
vaSpread1->SetRow2(4);
vaSpread1->SetCol2(3);
vaSpread1->SetBlockMode(TRUE);
// Define the background color for the cells
// red, RGB(255,0,0)
vaSpread1->SetBackColor(0x000000FF);
// Turn block mode off
vaSpread1->SetBlockMode(FALSE);
```

Visual Basic

```
' Select a block of cells
vaSpread1.Row = 2
vaSpread1.Col = 2
vaSpread1.Row2 = 4
vaSpread1.Col2 = 3
vaSpread1.BlockMode = True
' Define the background color for the cells
' red, RGB(255,0,0)
vaSpread1.BackColor = &H000000FF&
' Turn block mode off
vaSpread1.BlockMode = False
```

**See Also**

[Action](#), [BlockMode](#), [Clip](#), [Col](#), [Col2](#), [Row](#) properties



## RowHeaderDisplay Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the row header column displays letters or numbers or is blank.

### Syntax

C++ (OCX)      **long** CSpreadSheet::GetRowHeaderDisplay( );  
                  **void** CSpreadSheet::SetRowHeaderDisplay(**long** value);

C++ (VBX)      **short** CSpreadSheet::GetRowHeaderDisplay( );  
                  **void** CSpreadSheet::SetRowHeaderDisplay(**short** value);

Visual Basic    [form.]Spread.RowHeaderDisplay[ = setting%]

### OCX Use

Style drop-down list box under Row Headers on the [Headers](#) property page

### Remarks

The following settings are available:

Setting	Description	Constant
0 - Blank	Displays blanks in the row header column	SS_HEADER_BLANK
1 - Display Numbers	(Default) Displays numbers in the row header column	SS_HEADER_NUMBERS
2 - Display Letters	Displays letters in the row header column	SS_HEADER_LETTERS

**Note** Use the RowHeaderDisplay property with row header cells that do not contain data. If you place data into a row header cell using the [Text](#) or [Value](#) properties, the value of the Text or Value property overrides any previously applied RowHeaderDisplay property setting.

### Data Type

Integer (Enumerated)

[Print](#)

[Copy](#)

[Close](#)

The following example sets the row header column to display letters.

C++

```
// Set row header display type  
vaSpread1->SetRowHeaderDisplay(SS_HEADER_LETTERS);
```

Visual Basic

```
' Set row header display type  
vaSpread1.RowHeaderDisplay = SS_HEADER_LETTERS
```

**See Also**

[ColHeaderDisplay](#), [Text](#), [Value](#) properties

---

## RowHeight Property

[See Also](#)

[Example](#)

### Description

Sets or returns the height of a specified row. This property is available at run time only.

### Syntax

C++ (OCX)	<b>double</b> CSpreadSheet::GetRowHeight( <b>long</b> <i>Index</i> ); <b>void</b> CSpreadSheet::SetRowHeight( <b>long</b> <i>Index</i> , <b>double</b> <i>value</i> );
C++ (VBX)	<b>float</b> CSpreadSheet::GetRowHeight( <b>short</b> <i>Index</i> ); <b>void</b> CSpreadSheet::SetRowHeight( <b>short</b> <i>Index</i> , <b>float</b> <i>value</i> );
Visual Basic	[ <i>form.</i> ]Spread.RowHeight( <i>Index</i> )[ = <i>value!</i> ]

### Remarks

The RowHeight property is based on the system fixed font size. If the [UnitType](#) property is set to 2 (Twips), you must specify the height in twips.

Specify the row using the *Index* parameter.

### Data Type

VBX: Single Array  
OCX: Double

[Print](#)

[Copy](#)

[Close](#)

The following example sets the height of row 3.

C++

```
// Set the height of a selected row  
vaSpread1->SetRowHeight(3,19.6);
```

Visual Basic

```
' Set the height of a selected row  
vaSpread1.RowHeight(3) = 19.6
```

**See Also**

[ColWidth](#), [UnitType](#) properties

[RowHeightChange](#) event

[RowHeightToTwips](#) function

---

## RowHidden Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether a row is hidden. This property is available at run time only.

### Syntax

C++                    **BOOL** CSpreadSheet::GetRowHidden( );  
                         **void** CSpreadSheet::SetRowHidden(**BOOL** value);

Visual Basic        [form.]Spread.RowHidden[ = *boolean%*]

### Remarks

Before using the RowHidden property, set the [Row](#) property to specify which row to hide.

The default value for the RowHidden property is False.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example hides row 2.

**C++**

```
// Select a row  
vaSpread1->SetRow(2);  
// Hide the current row  
vaSpread1->SetRowHidden(TRUE);
```

**Visual Basic**

```
' Select a row  
vaSpread1.Row = 2  
' Hide the current row  
vaSpread1.RowHidden = True
```



**See Also**

[ColHidden](#), [Row](#) properties

---

## RowPageBreak Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether a page break occurs at the specified row when printing the spreadsheet. This property is available at run time only.

### Syntax

C++                    **BOOL** CSpreadSheet::GetRowPageBreak( );  
                         **void** CSpreadSheet::SetRowPageBreak(**BOOL** value);

Visual Basic        [form.]Spread.RowPageBreak[ = *boolean%*]

### Remarks

The default value for the RowPageBreak property is False.

The page break occurs above the specified row. Specify the row with the [Row](#) property.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example sets a print page break above spreadsheet row 3.

C++

```
// Select a row
vaSpread1->SetRow(3);
// Set the selected row as a print page break
vaSpread1->SetRowPageBreak(TRUE);
```

Visual Basic

```
' Select a row
vaSpread1.Row = 3
' Set the selected row as a print page break
vaSpread1.RowPageBreak = True
```

**See Also**

[ColPageBreak](#), [Row](#) properties

---

## RowsFrozen Property

[See Also](#)

[Example](#)

### Description

Sets or returns the number of nonscrolling, frozen rows.

### Syntax

C++	<b>long</b> CSpreadSheet::GetRowsFrozen( ); <b>void</b> CSpreadSheet::SetRowsFrozen( <b>long</b> value);
Visual Basic	[form.]Spread.RowsFrozen[ = value&]

### OCX Use

Rows Frozen for Scrolling box under Row Headers on the [Headers](#) property page

### Remarks

The default value for the RowsFrozen property is 0, which means that only the column header row is frozen.

**Note** Setting the RowsFrozen property to a value other than 0 may change the [TopRow](#) property setting.

The RowsFrozen property specifies which rows in the spreadsheet do not scroll. The frozen rows are always the top  $n$  rows, where  $n$  is the value of this property.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example makes the first two rows frozen.

C++

```
// Set the top two rows not to scroll  
vaSpread1->SetRowsFrozen(2);
```

Visual Basic

```
' Set the top two rows not to scroll  
vaSpread1.RowsFrozen = 2
```

**See Also**

[ColsFrozen](#), [TopRow](#) properties

---

## ScrollBarExtMode Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether to display scroll bars at all times or only when needed.

### Syntax

C++                    **BOOL** CSpreadSheet::GetScrollBarExtMode( );  
                         **void** CSpreadSheet::SetScrollBarExtMode(**BOOL** value);

Visual Basic        [form.]Spread.ScrollBarExtMode[ = boolean%]

### OCX Use

Show Only if Needed check box under Scroll Bars on the [Display](#) property page

### Remarks

The default value for the ScrollBarExtMode property is False.

When the ScrollBarExtMode property is set to True, the application hides the scroll bars when the maximum number of rows or columns fit in the control boundaries. When the ScrollBarExtMode property is set to False, the application displays the scroll bars at all times.

The ScrollBarExtMode property is valid only if the [ScrollBars](#) property is set to a value other than 0 (None).

### Data Type

Integer (Boolean)



[Print](#)

[Copy](#)

[Close](#)

The following example displays the scroll bars only if needed.

C++

```
vaSpread1->SetScrollBarExtMode (TRUE) ;
```

Visual Basic

```
vaSpread1.ScrollBarExtMode = True
```

**See Also**

[ScrollBarMaxAlign](#), [ScrollBars](#), [ScrollBarShowMax](#) properties

---

## ScrollBarMaxAlign Property

[See Also](#)

[Example](#)

### Description

Sets or returns the alignment of the last row and column on the displayed portion of the spreadsheet.

### Syntax

C++                    **BOOL** CSpreadSheet::GetScrollBarMaxAlign( );  
                         **void** CSpreadSheet::SetScrollBarMaxAlign(**BOOL** value);

Visual Basic        [form.]Spread.ScrollBarMaxAlign[ = boolean%]

### OCX Use

Align at Last Row and Column check box under Scroll Bars on the [Display](#) property page

### Remarks

The default value for the ScrollBarMaxAlign property is True.

When the ScrollBarMaxAlign property is set to True, the last row and column stop scrolling at the bottom and right of the displayed portion of the spreadsheet. When the ScrollBarMaxAlign property is set to False, the spreadsheet continues to scroll until the last row is at the top or the last column is at the left side of the displayed portion of the spreadsheet.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example specifies that the last row and column stop scrolling at the bottom and right of the displayed portion of the spreadsheet.

C++

```
// Determine if the last row/col scrolls to the end of the screen  
vaSpread1->SetScrollBarMaxAlign(TRUE);
```

Visual Basic

```
' Determine if the last row/col scrolls to the end of the screen  
vaSpread1.ScrollBarMaxAlign = True
```

**See Also**

[ScrollBarExtMode](#), [ScrollBars](#), [ScrollBarShowMax](#) properties

## ScrollBars Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether and how the spreadsheet displays scroll bars.

### Syntax

C++ (OCX)      **long** CSpreadSheet::GetScrollBars( );  
                  **void** CSpreadSheet::SetScrollBars(**long** value);

C++ (VBX)      **short** CSpreadSheet::GetScrollBars( );  
                  **void** CSpreadSheet::SetScrollBars(**short** value);

Visual Basic    [form.]Spread.ScrollBars[ = setting%]

### OCX Use

Display drop-down list box under Scroll Bars on the [Display](#) property page

### Remarks

The following settings are available:

Setting	Description	Constant
0 - None	Does not display scroll bars	SS_SCROLLBAR_NONE
1 - Horizontal	Displays horizontal scroll bar	SS_SCROLLBAR_H_ONLY
2 - Vertical	Displays vertical scroll bar	SS_SCROLLBAR_V_ONLY
3 - Both	(Default) Displays horizontal and vertical scroll bars	SS_SCROLLBAR_BOTH

### Data Type

Integer (Enumerated)

[Print](#)

[Copy](#)

[Close](#)

The following example displays the vertical scroll bar only.

C++

```
// Display vertical scroll bar only  
vaSpread1->SetScrollBars(SS_SCROLLBAR_V_ONLY);
```

Visual Basic

```
' Display vertical scroll bar only  
vaSpread1.ScrollBars = SS_SCROLLBAR_V_ONLY
```

**See Also**

[ScrollBarExtMode](#), [ScrollBarMaxAlign](#), [ScrollBarShowMax](#) properties



---

## ScrollBarShowMax Property

[See Also](#)

[Example](#)

### Description

Sets or returns the maximum range of the scroll bars displayed in the spreadsheet.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetScrollBarShowMax( ); <b>void</b> CSpreadSheet::SetScrollBarShowMax( <b>BOOL</b> value);
Visual Basic	[form.]Spread.ScrollBarShowMax[ = boolean%]

### OCX Use

Box Reflects Max Rows check box under Scroll Bars on the [Display](#) property page

### Remarks

The default value for the ScrollBarShowMax property is True.

When the ScrollBarShowMax property is set to True, the scroll bar range reflects the maximum number of rows and columns. For example, if the maximum number of rows is 500, the scroll bar range is 1 to 500.

When the ScrollBarShowMax property is set to False, the scroll bar range depends on the settings of the [DataColCnt](#) and [DataRowCnt](#) properties. For the vertical scroll bar, the range is 1 to either 20 or the DataRowCnt property value, whichever is greater. For the horizontal scroll bar, the range is 1 to either 8 or the DataColCnt property value, whichever is greater.

The ScrollBarShowMax property is valid only if the [ScrollBars](#) property is set to a value other than 0 (None).

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example sets the scroll bar range to reflect the maximum number of rows and columns in the spreadsheet.

C++

```
vaSpread1->SetScrollBarShowMax(TRUE);
```

Visual Basic

```
vaSpread1.ScrollBarShowMax = True
```

**See Also**

[DataColCnt](#), [DataRowCnt](#), [ScrollBarExtMode](#), [ScrollBarMaxAlign](#), [ScrollBars](#) properties

---

## SelBlockCol Property

[See Also](#)    [Example](#)

### Description

Returns the left column of a selected block of cells. This property is available at run time only.

### Syntax

C++                    **long** CSpreadSheet::GetSelBlockCol( );  
Visual Basic        [form.]Spread.SelBlockCol

### Remarks

A return value of -1 designates that an entire row is selected.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets the column width for each column in a block selection.

C++

```
if(vaSpread1->GetIsBlockSelected( ))
for(lcol = vaSpread1->GetSelBlockCol( );
lcol <= vaSpread1->GetSelBlockCol2( );
lcol++)
{
vaSpread1->SetColWidth(lcol,SS_Num);
}
```

Visual Basic

```
If vaSpread1.IsBlockSelected Then
For lcol = vaSpread1.SelBlockCol To vaSpread1.SelBlockCol2
vaSpread1.ColWidth(lcol) = SS_Num
Next lcol
End If
```

**See Also**

[Action](#) (2 - Select Block), [IsBlockSelected](#), [SelBlockCol2](#), [SelBlockRow](#), [SelBlockRow2](#) properties  
[BlockSelected](#) event

## — SelBlockCol2 Property

[See Also](#)    [Example](#)

### Description

Returns the right column of a selected block of cells. This property is available at run time only.

### Syntax

C++                    **long** CSpreadSheet::GetSelBlockCol2( );  
Visual Basic        [form.]Spread.SelBlockCol2

### Remarks

A return value of —1 designates that an entire row is selected.

### Data Type

Long Integer

**See Also**

[Action](#) (2 - Select Block), [IsBlockSelected](#), [SelBlockCol](#), [SelBlockRow](#), [SelBlockRow2](#) properties  
[BlockSelected](#) event



## — SelBlockRow Property

[See Also](#)

[Example](#)

### Description

Returns the top row of a selected block of cells. This property is available at run time only.

### Syntax

C++                    **long** CSpreadSheet::GetSelBlockRow( );

Visual Basic        [form.]Spread.SelBlockRow

### Remarks

A return value of —1 designates that an entire column is selected.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets the row height for each row within the block selection.

C++

```
if(vaSpread1->GetIsBlockSelected( ))
for(lrow = vaSpread1->GetSelBlockRow( );
  lrow <= vaSpread1->GetSelBlockRow2( );
  lrow++)
{
  vaSpread1->SetRowHeight(lrow,SS_Num);
}
```

Visual Basic

```
If vaSpread1.IsBlockSelected Then
  For lrow = vaSpread1.SelBlockRow To vaSpread1.SelBlockRow2
    vaSpread1.RowHeight(lrow) = SS_Num
  Next lrow
End If
```

**See Also**

[Action](#) (2 - Select Block), [IsBlockSelected](#), [SelBlockCol](#), [SelBlockCol2](#), [SelBlockRow2](#) properties  
[BlockSelected](#) event

## — SelBlockRow2 Property

[See Also](#)

[Example](#)

### Description

Returns the bottom row of a selected block of cells. This property is available at run time only.

### Syntax

C++                    **long** CSpreadSheet::GetSelBlockRow2( );

Visual Basic        [form.]Spread.SelBlockRow2

### Remarks

A return value of —1 designates that an entire column is selected.

### Data Type

Long Integer

**See Also**

[Action](#) (2 - Select Block), [IsBlockSelected](#), [SelBlockCol](#), [SelBlockCol2](#), [SelBlockRow](#) properties  
[BlockSelected](#) event

## SelectBlockOptions Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the user can select columns, rows, blocks of cells, all cells, or any combination of these.

### Syntax

```
C++          short CSpreadSheet::GetSelectBlockOptions( );  
            void CSpreadSheet::SetSelectBlockOptions(short value);  
  
Visual Basic  [form.]Spread.SelectBlockOptions[ = value%]
```

### OCX Use

Columns, Rows, Blocks, and All check boxes under Select Block Options on the [Op\\_Mode](#) property page

### Remarks

The following values are available. Note that you can combine values 1, 2, 4, and 8 using the Or operator to limit user selections within the spreadsheet. The default value is 15 (combination of all values), which lets the user select columns, rows, blocks of cells, or the entire spreadsheet. Setting the SelectBlockOptions property to 0 prevents the user from making selections within the spreadsheet.

Value	Description	Constant
1	Lets the user select columns	SS_SELBLOCKOPT_COLS
2	Lets the user select rows	SS_SELBLOCKOPT_ROWS
4	Lets the user select blocks of cells	SS_SELBLOCKOPT_BLOCKS
8	Lets the user select the entire spreadsheet by clicking the upper-left cell	SS_SELBLOCKOPT_ALL

**Note** Setting the [EditModePermanent](#) property to True automatically sets the SelectBlockOptions property to 0. If you then set the EditModePermanent property to False, you must reset the SelectBlockOptions property to its original setting.

### Data Type

Integer

[Print](#)

[Copy](#)

[Close](#)

The following example lets the user select both rows and blocks of cells.

C++

```
// Allow row or block selection  
vaSpread1->SetSelectBlockOptions(SS_SELBLOCKOPT_ROWS|SS_SELBLOCKOPT_BLOCKS);
```

Visual Basic

```
' Allow row or block selection  
vaSpread1.SelectBlockOptions = SS_SELBLOCKOPT_ROWS OR SS_SELBLOCKOPT_BLOCKS
```

**See Also**

[EditModePermanent](#) property

[BlockSelected](#), [SelChange](#) events



---

## SellLength Property

[See Also](#)

[Example](#)

### Description

Sets or returns the length of text that can be selected within the active cell when the cell is in edit mode. This property is available at run time only.

### Syntax

C++                    **long** CSpreadSheet::GetSellLength( );  
                         **void** CSpreadSheet::SetSellLength(**long** value);

Visual Basic        [form.]Spread.SellLength[ = value&]

### Remarks

This property is valid only when the cell is in edit mode, that is, when the edit control is over the active cell and the cell has the focus.

The SellLength property cannot be used in the [EditMode](#) event.

### Data Type

Long Integer

**See Also**

[SelStart](#), [SelText](#) properties

---

## SelModelIndex Property

[See Also](#)

[Example](#)

### Description

Sets or returns the current selection when the spreadsheet is in single-selection operation mode. This property is available at run time only.

### Syntax

C++	<b>long</b> CSpreadSheet::GetSelModelIndex( ); <b>void</b> CSpreadSheet::SetSelModelIndex( <b>long</b> value);
Visual Basic	[form.]Spread.SelModelIndex[ = value&]

### Remarks

The value assigned or retrieved by the SelModelIndex property is a valid row number.

The default value for the SelModelIndex property is 1.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example selects row 5 of the spreadsheet.

C++

```
// Select row 5 if the current selection is not row 5
if(vaSpread1->GetSelModeIndex( ) != 5)
    vaSpread1->SetSelModeIndex(5);
```

Visual Basic

```
' Select row 5 if the current selection is not row 5
If vaSpread1.SelModeIndex <> 5 Then
    vaSpread1.SelModeIndex = 5
End If
```

**See Also**

[SelModeSelCount](#), [SelModeSelected](#) properties

---

## SelModeSelCount Property

[See Also](#)

[Example](#)

### Description

Returns the number of selected rows when the spreadsheet is in extended- or multiple-selection operation mode. This property is available at run time only.

### Syntax

C++ **long** CSpreadSheet::GetSelModeSelCount( );

Visual Basic `[form.]Spread.SelModeSelCount`

### Remarks

Use the [SelModelIndex](#) property when working in single-selection operation mode.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example retrieves the number of selected rows in the spreadsheet.

C++

```
x = vaSpread1->GetSelModeSelCount( );
```

Visual Basic

```
x = vaSpread1.SelModeSelCount
```

**See Also**

[Action](#) (17 - Add MultiSel Blocks, 18 - Get MultiSel Blocks), [SelModelIndex](#), [SelModeSelected](#) properties



---

## SelModeSelected Property

[See Also](#)

[Example](#)

### Description

Sets or returns the selection state of a row when the spreadsheet is in extended- or multiple-selection operation mode. This property is available at run time only.

### Syntax

C++ **BOOL** CSpreadSheet::GetSelModeSelected( );  
**void** CSpreadSheet::SetSelModeSelected(**BOOL** value);

Visual Basic [form.]Spread.SelModeSelected[ = boolean%]

### Remarks

Set the SelModeSelected property to True to select the row designated by the [Row](#) property. Set the Row property before using the SelModeSelected property.

Use the [SelModelIndex](#) property when working in single-selection operation mode.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example selects the first five rows of the spreadsheet.

**C++**

```
// Select the first 5 rows
for(x=1; x<=5; x++)
{
    vaSpread1->SetRow(x);
    vaSpread1->SetSelModeSelected(TRUE);
}
```

**Visual Basic**

```
' Select the first 5 rows
For x=1 to 5
    vaSpread1.Row = x
    vaSpread1.SelModeSelected = True
Next x
```

**See Also**

[Action](#) (17 - Add MultiSel Blocks, 18 - Get MultiSel Blocks), [OperationMode](#), [Row](#), [SelModeIndex](#), [SelModeSelCount](#) properties

---

## SelStart Property

[See Also](#)    [Example](#)

### Description

Sets or returns the starting location of selected text within the active cell. This property is available at run time only.

### Syntax

C++	<b>long</b> CSpreadSheet::GetSelStart( ); <b>void</b> CSpreadSheet::SetSelStart( <b>long</b> value);
Visual Basic	[form.]Spread.SelStart[ = value&]

### Remarks

This property is valid only when the cell is in edit mode, that is, when the edit control is over the active cell and the cell has the focus.

The SelStart property cannot be used in the [EditMode](#) event.

### Data Type

Long Integer

**See Also**

[SelLength](#), [SelText](#) properties

---

## SelText Property

[See Also](#)    [Example](#)

### Description

Sets or returns the selected text within the active cell. This property is available at run time only.

### Syntax

C++ (OCX)	<b>CString</b> CSpreadSheet::GetSelText( ); <b>void</b> CSpreadSheet::SetSelText( <b>LPCSTR</b> value);
C++ (MFC VBX)	<b>CString</b> CSpreadSheet::GetSelText( ); <b>void</b> CSpreadSheet::SetSelText( <b>LPCSTR</b> value);
C++ (OWL VBX)	<b>string</b> CSpreadSheet::GetSelText( ); <b>void</b> CSpreadSheet::SetSelText( <b>LPCSTR</b> value);
Visual Basic	[form.]Spread.SelText[ = text\$]

### Remarks

This property is valid only when the cell is in edit mode, that is, when the edit control is over the active cell and the cell has the focus.

The SelText property cannot be used in the [EditMode](#) event.

### Data Type

String

[Print](#)

[Copy](#)

[Close](#)

The following example replaces the second and third characters of the selected text with X's, then concatenates the text "Hello" to the selected text.

C++

```
CString buf;
vaSpread1->SetSelStart(2);
vaSpread1->SetSelLength(2);
vaSpread1->SetSelText("XX");
// Concatenate the word "Hello" to the currently selected text
buf = vaSpread1->GetSelText( );
buf+="Hello";
vaSpread1->SetSelText(buf);
```

Visual Basic

```
vaSpread1.SelStart = 2
vaSpread1.SelLength = 2
vaSpread1.SelText = "XX"
' Concatenate the word "Hello" to the currently selected text
vaSpread1.SelText = vaSpread1.SelText + "Hello"
```

**See Also**

[SelLength](#), [SelStart](#) properties



---

## ShadowColor Property

[See Also](#)

[Example](#)

### Description

Sets or returns the background shadow color of the row and column headers.

### Syntax

C++ (OCX)	<b>unsigned long</b> CSpreadSheet::GetShadowColor( ); <b>void</b> CSpreadSheet::SetShadowColor( <b>unsigned long</b> value);
C++ (VBX)	<b>COLORREF</b> CSpreadSheet::GetShadowColor( ); <b>void</b> CSpreadSheet::SetShadowColor( <b>COLORREF</b> value);
Visual Basic	[form.]Spread.ShadowColor[ = color]

### OCX Use

Select ShadowColor from the Property Name drop-down list box on the [Colors](#) property page.

### Remarks

The default value for the ShadowColor property is &H00C0C0C0& (light gray).

### Data Type

VBX: Color

OCX: Unsigned Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets the row and column headers to red.

C++

```
// Set the background color for the row and column headers
// red, RGB(255,0,0)
vaSpread1->SetShadowColor(0x000000FF);
```

Visual Basic

```
' Set the background color for the row and column headers
' red, RGB(255,0,0)
vaSpread1.ShadowColor = &H000000FF&
```

**See Also**

[ShadowDark](#), [ShadowText](#) properties

## ShadowDark Property

[See Also](#)

[Example](#)

### Description

Sets or returns the color of the bottom and right borders of the row and column headers.

### Syntax

C++ (OCX)	<b>unsigned long</b> CSpreadSheet::GetShadowDark( ); <b>void</b> CSpreadSheet::SetShadowDark( <b>unsigned long value</b> );
C++ (VBX)	<b>COLORREF</b> CSpreadSheet::GetShadowDark( ); <b>void</b> CSpreadSheet::SetShadowDark( <b>COLORREF value</b> );
Visual Basic	[ <i>form.</i> ]Spread.ShadowDark[ = <i>color</i> ]

### OCX Use

Select ShadowDark from the Property Name drop-down list box on the [Colors](#) property page.

### Remarks

The default value for the ShadowDark property is &H00808080& (dark gray).

### Data Type

VBX: Color

OCX: Unsigned Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets the bottom- and right-border color to blue.

C++

```
// Set the outline color for the row and column headers
// blue, RGB(0,0,255)
vaSpread1->SetShadowDark(0x00FF0000);
```

Visual Basic

```
' Set the outline color for the row and column headers
' blue, RGB(0,0,255)
vaSpread1.ShadowDark = &H00FF0000&
```

**See Also**

[ShadowColor](#), [ShadowText](#) properties

---

## ShadowText Property

[See Also](#)

[Example](#)

### Description

Sets or returns the text color used for the row and column headers.

### Syntax

C++ (OCX)	<b>unsigned long</b> CSpreadSheet::GetShadowText( ); <b>void</b> CSpreadSheet::SetShadowText( <b>unsigned long</b> value);
C++ (VBX)	<b>COLORREF</b> CSpreadSheet::GetShadowText( ); <b>void</b> CSpreadSheet::SetShadowText( <b>COLORREF</b> value);
Visual Basic	[form.]Spread.ShadowText[ = color]

### OCX Use

Select ShadowText from the Property Name drop-down list box on the [Colors](#) property page.

### Remarks

The default value for the ShadowText property is &H00000000& (black).

### Data Type

VBX: Color

OCX: Unsigned Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets the row and column header color to green.

C++

```
// Set the text color for the row and column headers  
// green, RGB(0,255,0)  
vaSpread1->SetShadowText(0x0000FF00);
```

Visual Basic

```
' Set the text color for the row and column headers  
' green, RGB(0,255,0)  
vaSpread1.ShadowText = &H0000FF00&
```



**See Also**

[ShadowColor](#), [ShadowDark](#) properties

## SortBy Property

[See Also](#)   [Example](#)

### Description

Sets or returns whether to sort by rows or by columns. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetSortBy( );  
                         **void** CSpreadSheet::SetSortBy(**short** value);

Visual Basic        [form.]Spread.SortBy[ = setting%]

### Remarks

The following settings are available:

Setting	Description	Constant
0 - Row	Sorts by rows	SS_SORT_BY_ROW
1 - Col	Sorts by columns	SS_SORT_BY_COL

Use the SortBy property with the [Action](#) property.

### Data Type

Integer (Enumerated)

[Print](#)

[Copy](#)

[Close](#)

The following example sorts the spreadsheet by rows.

C++

```
// Select a block of cells
vaSpread1->SetRow(1);
vaSpread1->SetCol(1);
vaSpread1->SetRow2(3);
vaSpread1->SetCol2(2);
vaSpread1->SetSortBy(SS_SORT_BY_ROW);
// Set sort definition for key 1
vaSpread1->SetSortKey(1,1);
vaSpread1->SetSortKeyOrder(1,SS_SORT_ORDER_ASCENDING);
// Set sort definition for key 2
vaSpread1->SetSortKey(2,2);
vaSpread1->SetSortKeyOrder(2,SS_SORT_ORDER_DESCENDING);
```

Visual Basic

```
' Select a block of cells
vaSpread1.Row = 1
vaSpread1.Col = 1
vaSpread1.Row2 = 3
vaSpread1.Col2 = 2
vaSpread1.SortBy = SS_SORT_BY_ROW
' Set sort definition for key 1
vaSpread1.SortKey(1) = 1
vaSpread1.SortKeyOrder(1) = SS_SORT_ORDER_ASCENDING
' Set sort definition for key 2
vaSpread1.SortKey(2) = 2
vaSpread1.SortKeyOrder(2) = SS_SORT_ORDER_DESCENDING
```

**See Also**

[Action](#) (25 - Sort), [SortKey](#), [SortKeyOrder](#) properties

---

## SortKey Property

[See Also](#)   [Example](#)

### Description

Sets or returns individualsort actions within the spreadsheet. This property is available at run time only.

### Syntax

C++	<b>long</b> CSpreadSheet::GetSortKey( <b>short</b> Index); <b>void</b> CSpreadSheet::SetSortKey( <b>short</b> Index, <b>long</b> value);
Visual Basic	[form.]Spread.SortKey(Index) [ = value&]

### Remarks

The index used in the code refers to one of the three possible sort keys. The index for the first sort key is 1.

Use the SortKey property with the [Action](#) property.

### Data Type

Long Integer

**See Also**

[Action](#) (25 - Sort), [SortBy](#), [SortKeyOrder](#) properties

## SortKeyOrder Property

[See Also](#)

[Example](#)

### Description

Sets or returns the type of sorting to perform on each sort key. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetSortKeyOrder(**short** Index);  
                         **void** CSpreadSheet::SetSortKeyOrder(**short** Index, **short** value);

Visual Basic        [form.]Spread.SortKeyOrder(Index)[ = setting%]

### Remarks

The index used in the code refers to one of the three possible sort keys. The index for the first sort key is 1.

The following settings are available:

Setting	Description	Constant
0 - None	Does not sort	SS_SORT_ORDER_NONE
1 - Ascending	Sorts in ascending order	SS_SORT_ORDER_ASCENDING
2 - Descending	Sorts in descending order	SS_SORT_ORDER_DESCENDING

Use the SortKeyOrder property with the [Action](#) property.

### Data Type

Integer (Enumerated)

**See Also**

[Action](#) (25 - Sort), [SortBy](#), [SortKey](#) properties



---

## StartingColNumber Property

[See Also](#)

[Example](#)

### Description

Sets or returns the column number of the first column to display.

### Syntax

C++	<b>long</b> CSpreadSheet::GetStartingColNumber( ); <b>void</b> CSpreadSheet::SetStartingColNumber( <b>long</b> value);
Visual Basic	[form.]Spread.StartingColNumber[ = value&]

### OCX Use

Starting Column Number box under Col Headers on the [Headers](#) property page

### Remarks

The default value for the StartingColNumber property is 1.

The StartingColNumber property changes the starting column header number for display purposes only. The spreadsheet displays subsequent column numbers sequentially.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets the starting column number to 5.

C++

```
// Set the starting column number  
vaSpread1->SetStartingColNumber(5);
```

Visual Basic

```
' Set the starting column number  
vaSpread1.StartingColNumber = 5
```

**See Also**

[StartingRowNumber](#) property

---

## StartingRowNumber Property

[See Also](#)

[Example](#)

### Description

Sets or returns the row number of the first row to display.

### Syntax

C++	<b>long</b> CSpreadSheet::GetStartingRowNumber( ); <b>void</b> CSpreadSheet::SetStartingRowNumber( <b>long value</b> );
Visual Basic	[ <i>form.</i> ]Spread.StartingRowNumber[ = <i>value</i> &]

### OCX Use

Starting Row Number box under Row Headers on the [Headers](#) property page

### Remarks

The default value for the StartingRowNumber property is 1.

The StartingRowNumber property changes the starting row header number for display purposes only. The spreadsheet displays subsequent row numbers sequentially.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets the starting row number to 5.

C++

```
// Set the starting row number  
vaSpread1->SetStartingRowNumber(5);
```

Visual Basic

```
' Set the starting row number  
vaSpread1.StartingRowNumber = 5
```

**See Also**

[StartingColNumber](#) property

---

## TabIndex Property

[See Also](#)

### Description

Sets or returns the tab order of a control within its parent form.

### Syntax

C++	<b>short</b> CSpreadSheet::GetTabIndex( ); <b>void</b> CSpreadSheet::SetTabIndex( <b>short</b> value);
Visual Basic	[form.]Spread.TabIndex = value%

### OCX Use

The TabIndex property is a standard extender property that can be provided by your container.

### Remarks

You can set the TabIndex property to be any integer from 0 to  $(n-1)$ , where  $n$  is the number of controls on the form that have a TabIndex property.

By default, the tab order of controls is determined by the order in which you put them on the form. Each new control is placed last in the tab order. If you change the value of a control's TabIndex property to adjust the default tab order, the tab orders for the other controls are updated.

All controls except menus and timers are included in the tab order. At run time, invisible or disabled controls and controls that cannot receive the focus remain in the tab order but are skipped during tabbing.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

Integer

**See Also**

[TabStop](#) property



---

## TabStop Property

[See Also](#)

### Description

Sets or returns whether the user can use the Tab key to move the focus to a control.

### Syntax

C++	<b>short</b> CSpreadSheet::GetTabStop( ); <b>void</b> CSpreadSheet::SetTabStop( <b>short</b> value);
Visual Basic	[form.]Spread.TabStop[ = <i>boolean</i> %]

### OCX Use

The TabStop property is a standard extender property that can be provided by your container.

### Remarks

The default value for the TabStop property is True.

Use this property to add a control to or remove a control from the tab order.

If the TabStop property is set to True, the control can receive the focus when the user presses the Tab key. If the TabStop property is set to False, the focus bypasses the control when the user presses the Tab key, although the control still holds its place in the actual tab order.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

Integer (Boolean)

**See Also**

[TabIndex](#) property

---

## Tag Property

### Description

Sets or returns extra data for your application.

### Syntax

C++                    **LPCSTR** CSpreadSheet::GetTag( );  
                         **void** CSpreadSheet::SetTag(**LPCSTR** value);

Visual Basic        [form.]Spread.Tag[ = text\$]

### OCX Use

The Tag property is a standard extender property that can be provided by your container.

### Remarks

The default value for the Tag property is an empty string.

Use this property to assign an identification string to a control without affecting any of its other property settings.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

String

## Text Property

[See Also](#)   [Example](#)

### Description

Sets or returns the contents of the specified cell. This property is available at run time only.

### Syntax

C++ (OCX)      **CString** CSpreadSheet::GetText( );  
                  **void** CSpreadSheet::SetText(**LPCSTR** value);

C++ (MFC VBX)    **CString** CSpreadSheet::GetText( );  
                  **void** CSpreadSheet::SetText(**LPCSTR** value);

C++ (OWL VBX)    **string** CSpreadSheet::GetText( );  
                  **void** CSpreadSheet::SetText(**LPCSTR** value);

Visual Basic      [form.]Spread.Text[ = text\$]

### Remarks

The following table describes the type of data that can be entered into each cell type using the Text property:

Cell Type	Type of Data
Date	Formatted text
Edit	Formatted text (Validation is performed. If the <a href="#">TypeEditMultiLine</a> property is set to True, a linefeed character (ASCII 10 or '\n' in C++) can be used to force a line break when setting text.)
Float	Formatted text
Integer	Formatted text
PIC	Formatted text
Static Text	Formatted text (If the <a href="#">TypeTextWordWrap</a> property is set to True, a linefeed character (ASCII 10 or '\n' in C++) can be used to force a line break when setting text.)
Time	Formatted text (When using an a.m./p.m. indicator, include a space between the time value and the indicator, for example, "10:00:00 p.m.")
Button	Integer (If a two-state button is used, the Text property sets or returns a 1 if the button is down or a 0 if the button is up.)
Combo Box	Formatted text (text value of the currently selected item)
Picture	Not used
Check Box	Integer (The Text property sets or returns a 1 if the check box is checked or a 0 if the check box is not checked.)
Owner-Drawn	String only

Validation is performed to ensure invalid data is not set into the specified cell.

Use the [Clip](#) property rather than the Text property for large amounts of data. Use the [Value](#) property to set or return unformatted data.

Set the [Col](#) and [Row](#) properties to specify a cell before using the Text property.

### Data Type

String

[Print](#)

[Copy](#)

[Close](#)

The following example sets text for the specified cell.

**C++**

```
CString buf[100];
// Select a cell
vaSpread1->SetRow(3);
vaSpread1->SetCol(2);
// Enter data into the cell
vaSpread1->SetText("Hello");
// Concatenate the word "Hello" to the contents of the cell
buf = vaSpread1->GetText( );
buf+="Hello";
vaSpread1->SetText(buf);
```

**Visual Basic**

```
' Select a cell
vaSpread1.Row = 3
vaSpread1.Col = 2
' Enter data into the cell
vaSpread1.Text = "Hello"
' Concatenate the word "Hello" to the contents of the cell
vaSpread1.Text = vaSpread1.Text + "Hello"
```

**See Also**

[CellType](#), [Clip](#), [ClipValue](#), [Col](#), [Row](#), [Value](#) properties

[GetText](#), [SetText](#) functions

---

## Top Property

[See Also](#)

### Description

Sets or returns the distance between the internal top edge of an object and the top edge of its container.

### Syntax

C++	<b>long</b> CSpreadSheet::GetTop( ); <b>void</b> CSpreadSheet::SetTop( <b>long</b> value);
Visual Basic	[form.]Spread.Top[ = value!]

### OCX Use

The Top property is a standard extender property that can be provided by your container.

### Remarks

The value for this property changes as the user or the application moves the control. The Top property is measured in units according to the coordinate system of its container.

In Visual Basic, the measurement unit used by the Top property depends on the setting of the form's ScaleMode property. The default ScaleMode setting is twips (1/1440 of an inch). C++ uses pixels as the measurement unit.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

Visual Basic: Single  
C++: Long

**See Also**

[Left](#) property



---

## TopRow Property

[See Also](#)   [Example](#)

### Description

Sets or returns the row that is currently displayed as the top scrollable row on the screen. This property is available at run time only.

### Syntax

C++	<b>long</b> CSpreadSheet::GetTopRow( ); <b>void</b> CSpreadSheet::SetTopRow( <b>long</b> value);
Visual Basic	[form.]Spread.TopRow[ = value&]

### Remarks

Setting the [RowsFrozen](#) property to a value other than 0 may change the TopRow property setting.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets the top row of the spreadsheet to 5.

C++

```
// Set display's top row  
vaSpread1->SetTopRow(5);
```

Visual Basic

```
' Set display's top row  
vaSpread1.TopRow = 5
```

**See Also**

[Action](#) (1 - Go To), [LeftCol](#), [RowsFrozen](#) properties

[TopLeftChange](#) event

[GetBottomRightCell](#) function

## TypeButtonAlign Property

[See Also](#)

[Example](#)

### Description

Sets or returns the method used to align the picture and text within a button cell. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetTypeButtonAlign( );  
                         **void** CSpreadSheet::SetTypeButtonAlign(**short value**);

Visual Basic        [form.]Spread.TypeButtonAlign[ = *setting%*]

### Remarks

Use the TypeButtonAlign property with button cells.

The following settings are available:

Setting	Description	Constant
0 - Text Bottom/Picture Top	(Default) Text at the bottom, picture at the top	SS_CELL_BUTTON_ALIGN_BOTTOM
1 - Text Top/Picture Bottom	Text at the top, picture at the bottom	SS_CELL_BUTTON_ALIGN_TOP
2 - Text Left/Picture Right	Text at the left, picture at the right	SS_CELL_BUTTON_ALIGN_LEFT
3 - Text Right/Picture Left	Text at the right, picture at the left	SS_CELL_BUTTON_ALIGN_RIGHT

If you use the TypeButtonAlign property to retrieve the alignment method for a block of cells that do not all use the same alignment method, the alignment method used in the active cell is returned.

### Data Type

Integer (Enumerated)

[Print](#)

[Copy](#)

[Close](#)

The following example left aligns text in a button cell.

C++

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Define cell as type BUTTON
vaSpread1->SetCellType(SS_CELL_TYPE_BUTTON);
// red, RGB(255,0,0)
vaSpread1->SetTypeButtonColor(0x000000FF);
vaSpread1->SetTypeButtonText("Hello");
// black, RGB(0,0,0)
vaSpread1->SetTypeButtonTextColor(0x00000000);
// medium gray, RGB(128,128,128)
vaSpread1->SetTypeButtonDarkColor(0x00808080);
// green, RGB(0,255,0)
vaSpread1->SetTypeButtonLightColor(0x0000FF00);
// black, RGB(0,0,0)
vaSpread1->SetTypeButtonBorderColor(0x00000000);
vaSpread1->SetTypeButtonShadowSize(2);
vaSpread1->SetTypeButtonType(SS_CELL_BUTTON_NORMAL);
vaSpread1->SetTypeButtonAlign(SS_CELL_BUTTON_ALIGN_LEFT);
```

Visual Basic

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Define cell as type BUTTON
vaSpread1.CellType = SS_CELL_TYPE_BUTTON
' red, RGB(255,0,0)
vaSpread1.TypeButtonColor = &H000000FF&
vaSpread1.TypeButtonText = "Hello"
' black, RGB(0,0,0)
vaSpread1.TypeButtonTextColor = &H00000000&
' medium gray, RGB(128,128,128)
vaSpread1.TypeButtonDarkColor = &H00C0C0C0&
' green, RGB(0,255,0)
vaSpread1.TypeButtonLightColor = &H0000FF00&
' black, RGB(0,0,0)
vaSpread1.TypeButtonBorderColor = &H00000000&
vaSpread1.TypeButtonShadowSize = 2
vaSpread1.TypeButtonType = SS_CELL_BUTTON_NORMAL
vaSpread1.TypeButtonAlign = SS_CELL_BUTTON_ALIGN_LEFT
```

**See Also**

[CellType](#), [TypeButtonBorderColor](#), [TypeButtonColor](#), [TypeButtonDarkColor](#), [TypeButtonLightColor](#), [TypeButtonPicture](#), [TypeButtonPictureDown](#), [TypeButtonShadowSize](#), [TypeButtonText](#), [TypeButtonTextColor](#), [TypeButtonType](#) properties

## TypeButtonBorderColor Property

[See Also](#)

[Example](#)

### Description

Sets or returns the border color of a button cell. This property is available at run time only.

### Syntax

C++ (OCX)	<b>unsigned long</b> CSpreadSheet::GetTypeButtonBorderColor( ); <b>void</b> CSpreadSheet::SetTypeButtonBorderColor( <b>unsigned long</b> value);
C++ (VBX)	<b>COLORREF</b> CSpreadSheet::GetTypeButtonBorderColor( ); <b>void</b> CSpreadSheet::SetTypeButtonBorderColor( <b>COLORREF</b> value);
Visual Basic	[form.]Spread.TypeButtonBorderColor[ = color]

### OCX Use

Select TypeButtonBorderColor from the Property Name drop-down list box on the [Colors](#) property page.

### Remarks

Use the TypeButtonBorderColor property with button cells.

The default value for the TypeButtonBorderColor property is &H00000000& (black).

If you use the TypeButtonBorderColor property to retrieve the border color for a block of cells that do not all have the same border color, the border color of the active cell is returned.

**Note** In Windows 95, the TypeButtonBorderColor property has no effect on the button appearance.

### Data Type

VBX: Color

OCX: Unsigned Long Integer

**See Also**

[CellType](#), [TypeButtonAlign](#), [TypeButtonColor](#), [TypeButtonDarkColor](#), [TypeButtonLightColor](#), [TypeButtonPicture](#), [TypeButtonPictureDown](#), [TypeButtonShadowSize](#), [TypeButtonText](#), [TypeButtonTextColor](#), [TypeButtonType](#) properties



---

## TypeButtonColor Property

[See Also](#)

[Example](#)

### Description

Sets or returns the face background color of a button cell. This property is available at run time only.

### Syntax

C++ (OCX)	<b>unsigned long</b> CSpreadSheet::GetTypeButtonColor( ); <b>void</b> CSpreadSheet::SetTypeButtonColor( <b>unsigned long</b> value);
C++ (VBX)	<b>COLORREF</b> CSpreadSheet::GetTypeButtonColor( ); <b>void</b> CSpreadSheet::SetTypeButtonColor( <b>COLORREF</b> value);
Visual Basic	[form.]Spread.TypeButtonColor[ = color]

### OCX Use

Select TypeButtonColor from the Property Name drop-down list box on the [Colors](#) property page.

### Remarks

Use the TypeButtonColor property with button cells.

The default value for the TypeButtonColor property depends on your operating environment. For the Windows 3.1 environment, the default value is COLOR\_BTNFACE. For the Windows 95 environment, the default value is COLOR\_3DFACE.

### Data Type

VBX: Color

OCX: Unsigned Long Integer

**See Also**

[CellType](#), [TypeButtonAlign](#), [TypeButtonBorderColor](#), [TypeButtonDarkColor](#), [TypeButtonLightColor](#), [TypeButtonPicture](#), [TypeButtonPictureDown](#), [TypeButtonShadowSize](#), [TypeButtonText](#), [TypeButtonTextColor](#), [TypeButtonType](#) properties

## TypeButtonDarkColor Property

[See Also](#)

[Example](#)

### Description

Sets or returns the color of the dark highlight area around a button cell. This property is available at run time only.

### Syntax

C++ (OCX)	<b>unsigned long</b> CSpreadSheet::GetTypeButtonDarkColor( ); <b>void</b> CSpreadSheet::SetTypeButtonDarkColor( <b>unsigned long</b> value);
C++ (VBX)	<b>COLORREF</b> CSpreadSheet::GetTypeButtonDarkColor( ); <b>void</b> CSpreadSheet::SetTypeButtonDarkColor( <b>COLORREF</b> value);
Visual Basic	[form.]Spread.TypeButtonDarkColor[ = color]

### OCX Use

Select TypeButtonDarkColor from the Property Name drop-down list box on the [Colors](#) property page.

### Remarks

Use the TypeButtonDarkColor property with button cells.

A button has a border around it with a dark edge on two sides and a light edge on two sides to create a three-dimensional effect. Use the TypeButtonDarkColor property to change the color of the dark edge.

The default value for the TypeButtonDarkColor property depends on your operating environment. For the Windows 3.1 environment, the default value is COLOR\_BTNSHADOW. For the Windows 95 environment, the default value is COLOR\_3DSHADOW.

### Data Type

VBX: Color

OCX: Unsigned Long Integer

**See Also**

[CellType](#), [TypeButtonAlign](#), [TypeButtonBorderColor](#), [TypeButtonColor](#), [TypeButtonLightColor](#), [TypeButtonPicture](#), [TypeButtonPictureDown](#), [TypeButtonShadowSize](#), [TypeButtonText](#), [TypeButtonTextColor](#), [TypeButtonType](#) properties

## TypeButtonLightColor Property

[See Also](#)

[Example](#)

### Description

Sets or returns the color of the light highlight area around a button cell. This property is available at run time only.

### Syntax

C++ (OCX)	<b>unsigned long</b> CSpreadSheet::GetTypeButtonLightColor( ); <b>void</b> CSpreadSheet::SetTypeButtonLightColor( <b>unsigned long</b> value);
C++ (VBX)	<b>COLORREF</b> CSpreadSheet::GetTypeButtonLightColor( ); <b>void</b> CSpreadSheet::SetTypeButtonLightColor( <b>COLORREF</b> value);
Visual Basic	[form.]Spread.TypeButtonLightColor[ = color]

### OCX Use

Select TypeButtonLightColor from the Property Name drop-down list box on the [Colors](#) property page.

### Remarks

Use the TypeButtonLightColor property with button cells.

A button has a border around it with a dark edge on two sides and a light edge on two sides to create a three-dimensional effect. Use the TypeButtonLightColor property to change the color of the light edge.

The default value for the TypeButtonLightColor property depends on your operating environment. For the Windows 3.1 environment, the default value is COLOR\_BTNHIGHLIGHT. For the Windows 95 environment, the default value is COLOR\_3DHILIGHT.

### Data Type

VBX: Color

OCX: Unsigned Long Integer

**See Also**

[CellType](#), [TypeButtonAlign](#), [TypeButtonBorderColor](#), [TypeButtonColor](#), [TypeButtonDarkColor](#), [TypeButtonPicture](#), [TypeButtonPictureDown](#), [TypeButtonShadowSize](#), [TypeButtonText](#), [TypeButtonTextColor](#), [TypeButtonType](#) properties

## TypeButtonPicture Property

[See Also](#)

[Example](#)

### Description

Sets or returns the picture displayed when a button is up. This property is available at run time only.

### Syntax

C++ (OCX)	<b>CPicture</b> CSpreadSheet::GetTypeButtonPicture( ); <b>void</b> CSpreadSheet::SetTypeButtonPicture( <b>LPDISPATCH</b> value);
C++ (VBX)	<b>HPIC</b> CSpreadSheet::GetTypeButtonPicture( ); <b>void</b> CSpreadSheet::SetTypeButtonPicture( <b>HPIC</b> value);
Visual Basic	[form.]Spread.TypeButtonPicture[ = picture]

### Remarks

Use the TypeButtonPicture property with button cells.

Text, pictures, or both can appear within a button. If you use both, use the [TypeButtonAlign](#) property to specify positioning. Pictures do not overlap the highlighted areas of the button drawn by the spreadsheet.

Button cells support bitmaps and icons.

### Data Type

Picture

**See Also**

[CellType](#), [TypeButtonAlign](#), [TypeButtonBorderColor](#), [TypeButtonColor](#), [TypeButtonDarkColor](#), [TypeButtonLightColor](#), [TypeButtonPictureDown](#), [TypeButtonShadowSize](#), [TypeButtonText](#), [TypeButtonTextColor](#), [TypeButtonType](#) properties



## TypeButtonPictureDown Property

[See Also](#)

[Example](#)

### Description

Sets or returns the picture displayed when a button is down. This property is available at run time only.

### Syntax

C++ (OCX)	<b>CPicture</b> CSpreadSheet::GetTypeButtonPictureDown( ); <b>void</b> CSpreadSheet::SetTypeButtonPictureDown( <b>LPDISPATCH</b> value);
C++ (VBX)	<b>HPIC</b> CSpreadSheet::GetTypeButtonPictureDown( ); <b>void</b> CSpreadSheet::SetTypeButtonPictureDown( <b>HPIC</b> value);
Visual Basic	[form.]Spread.TypeButtonPictureDown[ = picture]

### Remarks

Use the TypeButtonPictureDown property with button cells.

Text, pictures, or both can appear within a button. If you use both, use the [TypeButtonAlign](#) property to specify positioning. Pictures do not overlap the highlighted areas of the button drawn by the spreadsheet.

Button cells support bitmaps and icons.

### Data Type

Picture

**See Also**

[CellType](#), [TypeButtonAlign](#), [TypeButtonBorderColor](#), [TypeButtonColor](#), [TypeButtonDarkColor](#), [TypeButtonLightColor](#), [TypeButtonPicture](#), [TypeButtonShadowSize](#), [TypeButtonText](#), [TypeButtonTextColor](#), [TypeButtonType](#) properties

---

## TypeButtonShadowSize Property

[See Also](#)

[Example](#)

### Description

Sets or returns the size of the shadow of a button cell. This property is available at run time only.

### Syntax

C++	<b>short</b> CSpreadSheet::GetTypeButtonShadowSize( ); <b>void</b> CSpreadSheet::SetTypeButtonShadowSize( <b>short</b> value);
Visual Basic	[form.]Spread.TypeButtonShadowSize[ = value%]

### Remarks

Use the TypeButtonShadowSize property with button cells.

Specify the size in pixels. The default value for the TypeButtonShadowSize property is 2.

### Data Type

Integer

**See Also**

[CellType](#), [TypeButtonAlign](#), [TypeButtonBorderColor](#), [TypeButtonColor](#), [TypeButtonDarkColor](#), [TypeButtonLightColor](#), [TypeButtonPicture](#), [TypeButtonPictureDown](#), [TypeButtonText](#), [TypeButtonTextColor](#), [TypeButtonType](#) properties

---

## TypeButtonText Property

[See Also](#)

[Example](#)

### Description

Sets or returns text to display in a button cell. This property is available at run time only.

### Syntax

C++ (OCX)	<b>CString</b> CSpreadSheet::GetTypeButtonText( ); <b>void</b> CSpreadSheet::SetTypeButtonText( <b>LPCSTR</b> value);
C++ (MFC VBX)	<b>CString</b> CSpreadSheet::GetTypeButtonText( ); <b>void</b> CSpreadSheet::SetTypeButtonText( <b>LPCSTR</b> value);
C++ (OWL VBX)	<b>string</b> CSpreadSheet::GetTypeButtonText( ); <b>void</b> CSpreadSheet::SetTypeButtonText( <b>LPCSTR</b> value);
Visual Basic	[form.]Spread.TypeButtonText[ = text\$]

### Remarks

Use the TypeButtonText property with button cells.

Text, pictures, or both can appear in the specified button. If you use both, use the [TypeButtonAlign](#) property to specify positioning.

### Data Type

String

**See Also**

[CellType](#), [TypeButtonAlign](#), [TypeButtonBorderColor](#), [TypeButtonColor](#), [TypeButtonDarkColor](#), [TypeButtonLightColor](#), [TypeButtonPicture](#), [TypeButtonPictureDown](#), [TypeButtonShadowSize](#), [TypeButtonTextColor](#), [TypeButtonType](#) properties

---

## TypeButtonTextColor Property

[See Also](#)

[Example](#)

### Description

Sets or returns the color of the text in a button cell. This property is available at run time only.

### Syntax

C++ (OCX)	<b>unsigned long</b> CSpreadSheet::GetTypeButtonTextColor( ); <b>void</b> CSpreadSheet::SetTypeButtonTextColor( <b>unsigned long</b> value);
C++ (VBX)	<b>COLORREF</b> CSpreadSheet::GetTypeButtonTextColor( ); <b>void</b> CSpreadSheet::SetTypeButtonTextColor( <b>COLORREF</b> value);
Visual Basic	[form.]Spread.TypeButtonTextColor[ = color]

### OCX Use

Select TypeButtonTextColor from the Property Name drop-down list box on the [Colors](#) property page.

### Remarks

Use the TypeButtonTextColor property with button cells.

The default value for the TypeButtonTextColor property is &H00000000& (black).

### Data Type

VBX: Color

OCX: Unsigned Long Integer

**See Also**

[CellType](#), [TypeButtonAlign](#), [TypeButtonBorderColor](#), [TypeButtonColor](#), [TypeButtonDarkColor](#), [TypeButtonLightColor](#), [TypeButtonPicture](#), [TypeButtonPictureDown](#), [TypeButtonShadowSize](#), [TypeButtonText](#), [TypeButtonType](#) properties



## TypeButtonType Property

[See Also](#)

[Example](#)

### Description

Sets or returns the type of button to be used in a button cell. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetTypeButtonType( );  
                         **void** CSpreadSheet::SetTypeButtonType(**short value**);

Visual Basic        [form.]Spread.TypeButtonType[ = *setting%*]

### Remarks

Use the TypeButtonType property with button cells.

The following settings are available:

Setting	Description	Constant
0 - Normal	(Default) Button does not stay down	SS_CELL_BUTTON_NORMAL
1 - Two-State	Button stays down after user presses it	SS_CELL_BUTTON_TWO_STATE

If you use a two-state button, use the [Text](#) property to determine the current state. If the Text property has a value of 1, the button is down.

### Data Type

Integer (Enumerated)

**See Also**

[CellType](#), [Text](#), [TypeButtonAlign](#), [TypeButtonBorderColor](#), [TypeButtonColor](#), [TypeButtonDarkColor](#), [TypeButtonLightColor](#), [TypeButtonPicture](#), [TypeButtonPictureDown](#), [TypeButtonShadowSize](#), [TypeButtonText](#), [TypeButtonTextColor](#) properties

---

## TypeCheckCenter Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the check box is centered in a check box cell. This property is available at run time only.

### Syntax

C++                    **BOOL** CSpreadSheet::GetTypeCheckCenter( );  
                         **void** CSpreadSheet::SetTypeCheckCenter(**BOOL** value);

Visual Basic        [form.]Spread.TypeCheckCenter[ = *boolean%*]

### Remarks

Use the TypeCheckCenter property with check box cells.

The default value for the TypeCheckCenter property is False, which left aligns the check box in a check box cell.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example left aligns the check box in a three-state check box cell.

C++

```
// Select a cell
vaSpread1->SetCol(2);
vaSpread1->SetRow(2);
// Define cell as type CHECKBOX
vaSpread1->SetCellType(SS_CELL_TYPE_CHECKBOX);
vaSpread1->SetTypeCheckType(SS_CHECKBOX_THREE_STATE);
vaSpread1->SetTypeCheckText("Check It");
vaSpread1->SetTypeCheckCenter(FALSE);
vaSpread1->SetTypeCheckTextAlign(SS_CHECKBOX_TEXT_RIGHT);
// Convert hBitmap to hPic for the VBX using C++
PIC picture0;
HPIC hPic0;
HBITMAP hBitmap0;
PIC picture1;
HPIC hPic1;
HBITMAP hBitmap1;
PIC picture2;
HPIC hPic2;
HBITMAP hBitmap2;
PIC picture3;
HPIC hPic3;
HBITMAP hBitmap3;
PIC picture4;
HPIC hPic4;
HBITMAP hBitmap4;
PIC picture5;
HPIC hPic5;
HBITMAP hBitmap5;
hBitmap0=LoadBitmap(AfxGetInstanceHandle( ),"pict0");
picture0.picData.bmp.hbitmap0=hBitmap0;
picture0.picType=PICTURE_BITMAP;
hPic0=AfxSetPict(NULL,&picture0);
hBitmap1=LoadBitmap(AfxGetInstanceHandle( ),"pict1");
picture1.picData.bmp.hbitmap1=hBitmap1;
picture1.picType=PICTURE_BITMAP;
hPic1=AfxSetPict(NULL,&picture1);
hBitmap2=LoadBitmap(AfxGetInstanceHandle( ),"pict2");
picture2.picData.bmp.hbitmap2=hBitmap2;
picture2.picType=PICTURE_BITMAP;
hPic2=AfxSetPict(NULL,&picture2);
hBitmap3=LoadBitmap(AfxGetInstanceHandle( ),"pict3");
picture3.picData.bmp.hbitmap3=hBitmap3;
picture3.picType=PICTURE_BITMAP;
hPic3=AfxSetPict(NULL,&picture3);
hBitmap4=LoadBitmap(AfxGetInstanceHandle( ),"pict4");
picture4.picData.bmp.hbitmap4=hBitmap4;
picture4.picType=PICTURE_BITMAP;
hPic4=AfxSetPict(NULL,&picture4);
hBitmap5=LoadBitmap(AfxGetInstanceHandle( ),"pict5");
picture5.picData.bmp.hbitmap5=hBitmap5;
picture5.picType=PICTURE_BITMAP;
hPic5=AfxSetPict(NULL,&picture5);
vaSpread1->SetTypeCheckPicture(0,hPic0);
vaSpread1->SetTypeCheckPicture(1,hPic1);
vaSpread1->SetTypeCheckPicture(2,hPic2);
vaSpread1->SetTypeCheckPicture(3,hPic3);
vaSpread1->SetTypeCheckPicture(4,hPic4);
vaSpread1->SetTypeCheckPicture(5,hPic5);
```

## Visual Basic

```
' Select a cell
vaSpread1.Col = 2
vaSpread1.Row = 2
' Define cell as type CHECKBOX
vaSpread1.CellType = SS_CELL_TYPE_CHECKBOX
vaSpread1.TypeCheckType = SS_CHECKBOX_THREE_STATE
vaSpread1.TypeCheckText = "Check It"
vaSpread1.TypeCheckCenter = False
vaSpread1.TypeCheckTextAlign = SS_CHECKBOX_TEXT_RIGHT
vaSpread1.TypeCheckPicture(0) = LoadPicture("c:\pict0.bmp")
vaSpread1.TypeCheckPicture(1) = LoadPicture("c:\pict1.bmp")
vaSpread1.TypeCheckPicture(2) = LoadPicture("c:\pict2.bmp")
vaSpread1.TypeCheckPicture(3) = LoadPicture("c:\pict3.bmp")
vaSpread1.TypeCheckPicture(4) = LoadPicture("c:\pict4.bmp")
vaSpread1.TypeCheckPicture(5) = LoadPicture("c:\pict5.bmp")
```

**See Also**

[CellType](#), [TypeCheckPicture](#), [TypeCheckText](#), [TypeCheckTextAlign](#), [TypeCheckType](#) properties

## TypeCheckPicture Property

[See Also](#)

[Example](#)

### Description

Sets or returns the picture used for each state of a check box. This property is available at run time only.

### Syntax

C++ (OCX)      **CPicture** CSpreadSheet::GetTypeCheckPicture(**short** Index);  
                  **void** CSpreadSheet::SetTypeCheckPicture(**short** Index, **LPDISPATCH** value);

C++ (VBX)      **HPIC** CSpreadSheet::GetTypeCheckPicture(**short** Index);  
                  **void** CSpreadSheet::SetTypeCheckPicture(**short** Index, **HPIC** value);

Visual Basic    [form.]Spread.TypeCheckPicture(Index) [ = picture]

### Remarks

Use the TypeCheckPicture property with check box cells.

The following index values are available:

Index Value	Description
0	Specifies picture used when check box is cleared (False state)
1	Specifies picture used when check box is selected (True state)
2	Specifies picture used when check box is cleared (False state) and pressed
3	Specifies picture used when check box is selected (True state) and pressed
4	Specifies picture used when check box is grayed (Indeterminate state)
5	Specifies picture used when check box is grayed (Indeterminate state) and pressed

### Data Type

Picture

**See Also**

[CellType](#), [TypeCheckCenter](#), [TypeCheckText](#), [TypeCheckTextAlign](#), [TypeCheckType](#) properties



---

## TypeCheckText Property

[See Also](#)

[Example](#)

### Description

Sets or returns the text in a check box cell. This property is available at run time only.

### Syntax

C++ (OCX)	<b>CString</b> CSpreadSheet::GetTypeCheckText( ); <b>void</b> CSpreadSheet::SetTypeCheckText( <b>LPCSTR</b> value);
C++ (MFC VBX)	<b>CString</b> CSpreadSheet::GetTypeCheckText( ); <b>void</b> CSpreadSheet::SetTypeCheckText( <b>LPCSTR</b> value);
C++ (OWL VBX)	<b>string</b> CSpreadSheet::GetTypeCheckText( ); <b>void</b> CSpreadSheet::SetTypeCheckText( <b>LPCSTR</b> value);
Visual Basic	[form.]Spread.TypeCheckText[ = text\$]

### Remarks

Use the TypeCheckText property with check box cells.

### Data Type

String

**See Also**

[CellType](#), [TypeCheckCenter](#), [TypeCheckPicture](#), [TypeCheckTextAlign](#), [TypeCheckType](#) properties

## TypeCheckTextAlign Property

[See Also](#)

[Example](#)

### Description

Sets or returns the placement of text in a check box cell. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetTypeCheckTextAlign( );  
                         **void** CSpreadSheet::SetTypeCheckTextAlign(**short** value);

Visual Basic        [form.]Spread.TypeCheckTextAlign[ = setting%]

### Remarks

Use the TypeCheckTextAlign property with check box cells.

The following settings are available:

Setting	Description	Constant
0 - Left	Displays text left of the graphic	SS_CHECKBOX_TEXT_LEFT
1 - Right	(Default) Displays text right of the graphic	SS_CHECKBOX_TEXT_RIGHT

### Data Type

Integer (Enumerated)

**See Also**

[CellType](#), [TypeCheckCenter](#), [TypeCheckPicture](#), [TypeCheckText](#), [TypeCheckType](#) properties

## TypeCheckType Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether a check box is two-state or three-state. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetTypeCheckType( );  
                         **void** CSpreadSheet::SetTypeCheckType(**short value**);

Visual Basic        [form.]Spread.TypeCheckType[ = *setting%*]

### Remarks

Use the TypeCheckType property with check box cells.

The following parameters are available:

Setting	Description	Constant
0 - Two-state	(Default) Specifies that the check box is a two-state check box	SS_CELL_CHECKBOX_NORMAL
1 - Three-state	Specifies that the check box is a three-state check box	SS_CELL_CHECKBOX_THREE_STATE

Use the [TypeCheckPicture](#) property to specify the picture to display for the check box.

### Data Type

Integer (Enumerated)

**See Also**

[CellType](#), [TypeCheckCenter](#), [TypeCheckPicture](#), [TypeCheckText](#), [TypeCheckTextAlign](#) properties

---

## TypeComboBoxCount Property

[See Also](#)

[Example](#)

### Description

Returns the number of items in the combo box list. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetTypeComboBoxCount( );

Visual Basic        [form.]Spread.TypeComboBoxCount

### Remarks

Use the TypeComboBoxCount property with combo box cells.

Set the [Col](#) and [Row](#) properties to select a cell before using the TypeComboBoxCount property.

### Data Type

Integer

[Print](#)

[Copy](#)

[Close](#)

The following example retrieves the number of items in a combo box list and selects the second item in the list.

C++

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Define cell as type COMBOBOX
vaSpread1->SetCellType(SS_CELL_TYPE_COMBOBOX);
vaSpread1->SetTypeComboBoxList("Combo 1\tCombo 2\tCombo 3");
vaSpread1->SetTypeComboBoxEditable(False);
x = vaSpread1->GetTypeComboBoxCount( );
// Select the second item in the combo box
vaSpread1->SetTypeComboBoxCurSel(1);
```

Visual Basic

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Define cell as type COMBOBOX
vaSpread1.CellType = SS_CELL_TYPE_COMBOBOX
vaSpread1.TypeComboBoxList = "Combo 1" + Chr$(9) + "Combo 2" + Chr$(9) + "Combo 3"
vaSpread1.TypeComboBoxEditable = False
x = vaSpread1.TypeComboBoxCount
' Select the second item in the combo box
vaSpread1.TypeComboBoxCurSel = 1
```



**See Also**

[Action](#) (26 - Combo Clear, 27 - Combo Remove String), [CellType](#), [TypeComboBoxCurSel](#), [TypeComboBoxEditable](#), [TypeComboBoxIndex](#), [TypeComboBoxList](#), [TypeComboBoxString](#) properties

## TypeComboBoxCurSel Property

[See Also](#)

[Example](#)

### Description

Sets or returns the selection in the combo box list. This property is available at run time only.

### Syntax

C++	<b>short</b> CSpreadSheet::GetTypeComboBoxCurSel( ); <b>void</b> CSpreadSheet::SetTypeComboBoxCurSel( <b>short value</b> );
Visual Basic	[ <i>form.</i> ]Spread.TypeComboBoxCurSel[ = <i>value%</i> ]

### Remarks

Use the TypeComboBoxCurSel property with combo box cells. The first item in the combo box list has a TypeComboBoxCurSel property value of 0.

Set the [Col](#) and [Row](#) properties to select a cell before using the TypeComboBoxCurSel property.

### Data Type

Integer

**See Also**

[Action](#) (26 - Combo Clear, 27 - Combo Remove String), [CellType](#), [TypeComboBoxCount](#), [TypeComboBoxEditable](#), [TypeComboBoxIndex](#), [TypeComboBoxList](#), [TypeComboBoxString](#) properties

---

## TypeComboBoxEditable Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the user can edit the text in a combo box cell. This property is available at run time only.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetTypeComboBoxEditable( ); <b>void</b> CSpreadSheet::SetTypeComboBoxEditable( <b>BOOL</b> value);
Visual Basic	[form.]Spread.TypeComboBoxEditable[ = boolean%]

### Remarks

Use the TypeComboBoxEditable property with combo box cells.

The default value for the TypeComboBoxEditable property is False.

### Data Type

Integer (Boolean)

**See Also**

[Action](#) (26 - Combo Clear, 27 - Combo Remove String), [CellType](#), [TypeComboBoxCount](#), [TypeComboBoxCurSel](#), [TypeComboBoxIndex](#), [TypeComboBoxList](#), [TypeComboBoxString](#) properties

## TypeComboBoxIndex Property

[See Also](#)

[Example](#)

### Description

Sets or returns the combo box item on which to perform an operation. This property is available at run time only.

### Syntax

C++	<b>short</b> CSpreadSheet::GetTypeComboBoxIndex( ); <b>void</b> CSpreadSheet::SetTypeComboBoxIndex( <b>short</b> value);
Visual Basic	[form.]Spread.TypeComboBoxIndex[ = value%]

### Remarks

Use the TypeComboBoxIndex property with combo box cells. Set the TypeComboBoxIndex property before using the [TypeComboBoxString](#) property and the [Action](#) property.

When you insert a new string, set the TypeComboBoxIndex property to -1 to force the string to be added to the end of the list.

Set the [Col](#) and [Row](#) properties to select a cell before using the TypeComboBoxIndex property.

The default value for the TypeComboBoxIndex property is 0.

### Data Type

Integer

**See Also**

[Action](#) (26 - Combo Clear, 27 - Combo Remove String), [CellType](#), [TypeComboBoxCount](#), [TypeComboBoxCurSel](#), [TypeComboBoxEditable](#), [TypeComboBoxList](#), [TypeComboBoxString](#) properties

## TypeComboBoxList Property

[See Also](#)

[Example](#)

### Description

Sets or returns the list of text strings displayed in a combo box cell. This property is available at run time only.

### Syntax

C++ (OCX)	<b>CString</b> CSpreadSheet::GetTypeComboBoxList( ); <b>void</b> CSpreadSheet::SetTypeComboBoxList( <b>LPCWSTR</b> value);
C++ (MFC VBX)	<b>CString</b> CSpreadSheet::GetTypeComboBoxList( ); <b>void</b> CSpreadSheet::SetTypeComboBoxList( <b>LPCSTR</b> value);
C++ (OWL VBX)	<b>string</b> CSpreadSheet::GetTypeComboBoxList( ); <b>void</b> CSpreadSheet::SetTypeComboBoxList( <b>LPCSTR</b> value);
Visual Basic	[form.]Spread.TypeComboBoxList[ = text\$]

### Remarks

Use the TypeComboBoxList property with combo box cells. Separate each item in the string list with a tab character (ASCII 9 or '\t' in C++).

If you use the [Text](#) property, specify the text for the item. Use the [Value](#) property to set the index of the specific item.

### Data Type

String



**See Also**

[Action](#) (26 - Combo Clear, 27 - Combo Remove String), [CellType](#), [TypeComboBoxCount](#), [TypeComboBoxCurSel](#), [TypeComboBoxEditable](#), [TypeComboBoxIndex](#), [TypeComboBoxString](#) properties

## TypeComboBoxString Property

[See Also](#)

[Example](#)

### Description

Sets or returns a string in a combo box list. This property is available at run time only.

### Syntax

C++ (OCX)	<b>CString</b> CSpreadSheet::GetTypeComboBoxString( ); <b>void</b> CSpreadSheet::SetTypeComboBoxString( <b>LPCWSTR</b> value);
C++ (MFC VBX)	<b>CString</b> CSpreadSheet::GetTypeComboBoxString( ); <b>void</b> CSpreadSheet::SetTypeComboBoxString( <b>LPCSTR</b> value);
C++ (OWL VBX)	<b>string</b> CSpreadSheet::GetTypeComboBoxString( ); <b>void</b> CSpreadSheet::SetTypeComboBoxString( <b>LPCSTR</b> value);
Visual Basic	[form.]Spread.TypeComboBoxString[ = text\$]

### Remarks

Use the TypeComboBoxString property with combo box cells.

Use the TypeComboBoxString property to retrieve a string from a combo box list or to insert a new string into the list.

When you insert a new string, set the [TypeComboBoxIndex](#) property to -1 to force the string to be added to the end of the list.

Set the [Col](#) and [Row](#) properties to select a cell before using the TypeComboBoxString property. Also, set the TypeComboBoxIndex property to specify which item you want from the list.

### Data Type

String

**See Also**

[Action](#) (26 - Combo Clear, 27 - Combo Remove String), [CellType](#), [TypeComboBoxCount](#), [TypeComboBoxCurSel](#), [TypeComboBoxEditable](#), [TypeComboBoxIndex](#), [TypeComboBoxList](#) properties

---

## TypeDateCentury Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether to display the year in century format in a date cell. This property is available at run time only.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetTypeDateCentury( ); <b>void</b> CSpreadSheet::SetTypeDateCentury( <b>BOOL</b> value);
Visual Basic	[form.]Spread.TypeDateCentury[ = boolean%]

### Remarks

Use the TypeDateCentury property with date cells.

If the TypeDateCentury property is set to True, the application displays the year with full century notation. For example, the application displays "1928" instead of "28".

The default value for the TypeDateCentury property is False.

**Note** If the TypeDateCentury property is set to False, date cells interpret dates earlier than 01/01/60 as century 2000 dates. For example, the date value 12/31/59 would be interpreted as December 31, 2059.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example displays the year without the century digits.

C++

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Define cell as type DATE
vaSpread1->SetCellType(SS_CELL_TYPE_DATE);
vaSpread1->SetTypeDateCentury(FALSE);
vaSpread1->SetTypeHAlign(SS_CELL_H_ALIGN_LEFT);
vaSpread1->SetTypeSpin(FALSE);
vaSpread1->SetTypeDateFormat(SS_CELL_DATE_FORMAT_MMDDYY);
vaSpread1->SetTypeDateMin("01011900");
vaSpread1->SetTypeDateMax("12312100");
```

Visual Basic

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Define cell as type DATE
vaSpread1.CellType = SS_CELL_TYPE_DATE
vaSpread1.TypeDateCentury = False
vaSpread1.TypeHAlign = SS_CELL_H_ALIGN_LEFT
vaSpread1.TypeSpin = False
vaSpread1.TypeDateFormat = SS_CELL_DATE_FORMAT_MMDDYY
vaSpread1.TypeDateMin = "01011900"
vaSpread1.TypeDateMax = "12312100"
```

**See Also**

[CellType](#), [TypeDateFormat](#), [TypeDateMax](#), [TypeDateMin](#), [TypeDateSeparator](#), [TypeHAlign](#), [TypeSpin](#) properties

## TypeDateFormat Property

[See Also](#)

[Example](#)

### Description

Sets or returns the format used to display the date in a date cell. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetTypeDateFormat( );  
                         **void** CSpreadSheet::SetDateFormat(**short** value);

Visual Basic        [form.]Spread.TypeDateFormat[ = setting%]

### Remarks

Use the TypeDateFormat property with date cells.

The following settings are available:

Setting	Example	Constant
0 - DDMONYY	15/NOV/95	SS_CELL_DATE_FORMAT_DDMONYY
1 - DDMMYY	15/11/95	SS_CELL_DATE_FORMAT_DDMMYY
2 - MMDDYY	11/15/95	SS_CELL_DATE_FORMAT_MMDDYY
3 - YYMMDD	95/11/15	SS_CELL_DATE_FORMAT_YYMMDD

The default value for the TypeDateFormat property is the value in the Date Format setting in the Windows Control Panel.

**Note** The spreadsheet does not recognize changes made to the settings in the Windows Control Panel until you restart Visual Basic or your application (or perform any operation that unloads and reloads the current .VBX or .OCX file).

### Data Type

Integer (Enumerated)

**See Also**

[CellType](#), [TypeDateCentury](#), [TypeDateMax](#), [TypeDateMin](#), [TypeDateSeparator](#), [TypeHAlign](#), [TypeSpin](#) properties



## TypeDateMax Property

[See Also](#)

[Example](#)

### Description

Sets or returns the maximum date value allowed in a date cell. This property is available at run time only.

### Syntax

C++ (OCX)	<b>CString</b> CSpreadSheet::GetTypeDateMax( ); <b>void</b> CSpreadSheet::SetTypeDateMax( <b>LPCWSTR</b> value);
C++ (MFC VBX)	<b>CString</b> CSpreadSheet::GetTypeDateMax( ); <b>void</b> CSpreadSheet::SetTypeDateMax( <b>LPCSTR</b> value);
C++ (OWL VBX)	<b>string</b> CSpreadSheet::GetTypeDateMax( ); <b>void</b> CSpreadSheet::SetTypeDateMax( <b>LPCSTR</b> value);
Visual Basic	[form.]Spread.TypeDateMax[ = text\$]

### Remarks

Use the TypeDateMax property with date cells.

Use the format "MMDDYYYY" to set the TypeDateMax property.

The default value for the TypeDateMax property is "12312100".

### Data Type

String

**See Also**

[CellType](#), [TypeDateCentury](#), [TypeDateFormat](#), [TypeDateMin](#), [TypeDateSeparator](#), [TypeHAlign](#), [TypeSpin](#) properties

## TypeDateMin Property

[See Also](#)

[Example](#)

### Description

Sets or returns the minimum date value allowed in a date cell. This property is available at run time only.

### Syntax

C++ (OCX)	<b>CString</b> CSpreadSheet::GetTypeDateMin( ); <b>void</b> CSpreadSheet::SetTypeDateMin( <b>LPCTSTR</b> value);
C++ (MFC VBX)	<b>CString</b> CSpreadSheet::GetTypeDateMin( ); <b>void</b> CSpreadSheet::SetTypeDateMin( <b>LPCSTR</b> value);
C++ (OWL VBX)	<b>string</b> CSpreadSheet::GetTypeDateMin( ); <b>void</b> CSpreadSheet::SetTypeDateMin( <b>LPCSTR</b> value);
Visual Basic	[form.]Spread.TypeDateMin[ = text\$]

### Remarks

Use the TypeDateMin property with date cells.

Use the format "MMDDYYYY" to set the TypeDateMin property.

The default value for the TypeDateMin property is "01011990".

### Data Type

String

**See Also**

[CellType](#), [TypeDateCentury](#), [TypeDateFormat](#), [TypeDateMax](#), [TypeDateSeparator](#), [TypeHAlign](#), [TypeSpin](#) properties

---

## TypeDateSeparator Property

[See Also](#)

[Example](#)

### Description

Sets or returns the separator character used between the month, day, and year in a date cell. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetTypeDateSeparator( );  
                         **void** CSpreadSheet::SetTypeDateSeparator(**short value**);

Visual Basic        [form.]Spread.TypeDateSeparator[ = value%]

### Remarks

Use the TypeDateSeparator property with date cells.

The default value for the TypeDateSeparator property is the value in the Date Format setting in the Windows Control Panel.

**Note** The spreadsheet does not recognize changes made to the settings in the Windows Control Panel until you restart Visual Basic or your application (or perform any operation that unloads and reloads the current .VBX or .OCX file).

### Data Type

Integer

**See Also**

[CellType](#), [TypeDateCentury](#), [TypeDateFormat](#), [TypeDateMax](#), [TypeDateMin](#), [TypeHAlign](#), [TypeSpin](#) properties

## TypeEditCharCase Property

[See Also](#)

[Example](#)

### Description

Sets or returns the case of the characters typed into an edit cell. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetTypeEditCharCase( );  
                         **void** CSpreadSheet::SetTypeEditCharCase(**short** value);

Visual Basic        [form.]Spread.TypeEditCharCase[ = setting%]

### Remarks

Use the TypeEditCharCase property with edit cells.

The following settings are available:

Setting	Description	Constant
0 - Lower Case	Characters appear in lower case	SS_CELL_EDIT_CASE_LOWER_CASE
1 - No Case	(Default) Characters appear as typed by the user	SS_CELL_EDIT_CASE_NO_CASE
2 - Upper Case	Characters appear in upper case	SS_CELL_EDIT_CASE_UPPER_CASE

### Data Type

Integer (Enumerated)

[Print](#)

[Copy](#)

[Close](#)

The following example specifies that characters appear as typed by the user.

C++

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Define cell as type EDIT
vaSpread1->SetCellType(SS_CELL_TYPE_EDIT);
vaSpread1->SetTypeHAlign(SS_CELL_H_ALIGN_LEFT);
vaSpread1->SetTypeEditCharCase(SS_CELL_EDIT_CASE_NO_CASE);
vaSpread1->SetTypeEditCharSet(SS_CELL_EDIT_CHAR_SET_ASCII);
vaSpread1->SetTypeEditPassword(FALSE);
vaSpread1->SetTypeEditMultiLine(TRUE);
vaSpread1->SetTypeEditLen(60);
```

Visual Basic

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Define cell as type EDIT
vaSpread1.CellType = SS_CELL_TYPE_EDIT
vaSpread1.TypeHAlign = SS_CELL_H_ALIGN_LEFT
vaSpread1.TypeEditCharCase = SS_CELL_EDIT_CASE_NO_CASE
vaSpread1.TypeEditCharSet = SS_CELL_EDIT_CHAR_SET_ASCII
vaSpread1.TypeEditPassword = False
vaSpread1.TypeEditMultiLine = True
vaSpread1.TypeEditLen = 60
```



**See Also**

[CellType](#), [TypeEditCharSet](#), [TypeEditLen](#), [TypeEditMultiLine](#), [TypeEditPassword](#), [TypeHAlign](#) properties

## TypeEditCharSet Property

[See Also](#)

[Example](#)

### Description

Sets or returns the valid character set available for the user to type data into an edit cell. This property is available at run time only.

### Syntax

C++ **short** CSpreadSheet::GetTypeEditCharSet( );  
**void** CSpreadSheet::SetTypeEditCharSet(**short** value);

Visual Basic [form.]Spread.TypeEditCharSet[ = setting%]

### Remarks

Use the TypeEditCharSet property with edit cells.

The following settings are available:

Setting	Characters Accepted	Constant
0 - ASCII	(Default) Any ASCII character	SS_CELL_EDIT_CHAR_SET_ASCII
1 - Alpha	A—Z, a—z, space	SS_CELL_EDIT_CHAR_SET_ALPHA
2 - Alphanumeric	A—Z, a—z, 0—9, period, comma, space, minus sign (-)	SS_CELL_EDIT_CHAR_SET_ALPHANUMERIC
3 - Numeric	0—9, period, minus sign (-)	SS_CELL_EDIT_CHAR_SET_NUMERIC

### Data Type

Integer (Enumerated)

**See Also**

[CellType](#), [TypeEditCharCase](#), [TypeEditLen](#), [TypeEditMultiLine](#), [TypeEditPassword](#), [TypeHAlign](#) properties

## TypeEditLen Property

[See Also](#)

[Example](#)

### Description

Sets or returns the maximum number of characters allowed in an edit cell. This property is available at run time only.

### Syntax

C++ (OCX)	<b>long</b> CSpreadSheet::GetTypeEditLen( ); <b>void</b> CSpreadSheet::SetTypeEditLen( <b>long</b> value);
C++ (VBX)	<b>short</b> CSpreadSheet::GetTypeEditLen( ); <b>void</b> CSpreadSheet::SetTypeEditLen( <b>short</b> value);
Visual Basic	[form.]Spread.TypeEditLen[ = value&]

### Remarks

Use the TypeEditLen property with edit cells.

The default value for the TypeEditLen property is 60 characters.

A beep sounds to notify the user when the maximum number of characters is reached (unless the [NoBeep](#) property is set to True).

### Data Type

Integer

**See Also**

[CellType](#), [NoBeep](#), [TypeEditCharCase](#), [TypeEditCharSet](#), [TypeEditMultiLine](#), [TypeEditPassword](#), [TypeHAlign](#) properties

## TypeEditMultiLine Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the user can type multiple lines of data into an edit cell. This property is available at run time only.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetTypeEditMultiLine( ); <b>void</b> CSpreadSheet::SetTypeEditMultiLine( <b>BOOL</b> value);
Visual Basic	[form.]Spread.TypeEditMultiLine[ = boolean%]

### Remarks

Use the TypeEditMultiLine property with edit cells.

The default value for the TypeEditMultiLine property is False.

When editing a multiple-line edit cell, pressing the Enter key advances to the next line.

Use the [MaxTextRowHeight](#) property or the [MaxTextCellHeight](#) property in conjunction with the TypeEditMultiLine property to determine if the size of the cell needs adjusting to display all the text the user has typed.

### Data Type

Integer (Boolean)

**See Also**

[CellType](#), [MaxTextCellHeight](#), [MaxTextRowHeight](#), [TypeEditCharCase](#), [TypeEditCharSet](#), [TypeEditLen](#), [TypeEditPassword](#), [TypeHAlign](#) properties

---

## TypeEditPassword Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether a single-line edit cell displays asterisks (\*) instead of the characters typed by the user. This property is available at run time only.

### Syntax

C++ **BOOL** CSpreadSheet::GetTypeEditPassword( );  
**void** CSpreadSheet::SetTypeEditPassword(**BOOL** value);

Visual Basic [form.]Spread.TypeEditPassword[ = boolean%]

### Remarks

Use the TypeEditPassword property with single-line edit cells to let the user create a password for a cell.

When the TypeEditPassword property is set to True, each character typed by the user is displayed on the screen as an asterisk (\*). When the TypeEditPassword property is set to False, text is displayed as typed by the user.

The default value for the TypeEditPassword property is False.

### Data Type

Integer (Boolean)



**See Also**

[CellType](#), [TypeEditCharCase](#), [TypeEditCharSet](#), [TypeEditLen](#), [TypeEditMultiLine](#), [TypeHAlign](#) properties

---

## TypeFloatCurrencyChar Property

[See Also](#)

[Example](#)

### Description

Sets or returns the currency character used in a float cell. This property is available at run time only.

### Syntax

C++	<b>short</b> CSpreadSheet::GetTypeFloatCurrencyChar( ); <b>void</b> CSpreadSheet::SetTypeFloatCurrencyChar( <b>short</b> value);
Visual Basic	[form.]Spread.TypeFloatCurrencyChar[ = value%]

### Remarks

Use the TypeFloatCurrencyChar property with float cells.

You can use the ASCII value of the character to designate the currency character.

The default value for the TypeFloatCurrencyChar property is 0. If this property is set to 0, the spreadsheet uses the default currency character designated by the [FloatDefCurrencyChar](#) property. To override the FloatDefCurrencyChar property value for an individual float cell, set the TypeFloatCurrencyChar property to a value other than 0.

### Data Type

Integer

[Print](#)

[Copy](#)

[Close](#)

The following example designates a dollar sign (\$) as the currency character, a period as the decimal point character, and a comma as the separator character.

C++

```
// Select a cell
vaSpread1->SetCol(2);
vaSpread1->SetRow(2);
// Define cell as type FLOAT
vaSpread1->SetCellType(SS_CELL_TYPE_FLOAT);
vaSpread1->SetTypeFloatDecimalPlaces(2);
vaSpread1->SetTypeFloatMin("-9999999");
vaSpread1->SetTypeFloatMax("9999999");
vaSpread1->SetTypeFloatMoney(TRUE);
vaSpread1->SetTypeFloatSeparator(TRUE);
vaSpread1->SetTypeHAlign(SS_CELL_H_ALIGN_RIGHT);
vaSpread1->SetTypeFloatCurrencyChar('$');
vaSpread1->SetTypeFloatDecimalChar('.');
vaSpread1->SetTypeFloatSepChar(',');
```

Visual Basic

```
' Select a cell
vaSpread1.Col = 2
vaSpread1.Row = 2
' Define cell as type FLOAT
vaSpread1.CellType = SS_CELL_TYPE_FLOAT
vaSpread1.TypeFloatDecimalPlaces = 2
vaSpread1.TypeFloatMin = "-9999999"
vaSpread1.TypeFloatMax = "9999999"
vaSpread1.TypeFloatMoney = True
vaSpread1.TypeFloatSeparator = True
vaSpread1.TypeHAlign = SS_CELL_H_ALIGN_RIGHT
vaSpread1.TypeFloatCurrencyChar = ASC("$")
vaSpread1.TypeFloatDecimalChar = ASC(".")
vaSpread1.TypeFloatSepChar = ASC(",")
```

**See Also**

[CellType](#), [FloatDefCurrencyChar](#), [TypeFloatDecimalChar](#), [TypeFloatDecimalPlaces](#), [TypeFloatMax](#), [TypeFloatMin](#), [TypeFloatMoney](#), [TypeFloatSeparator](#), [TypeFloatSepChar](#), [TypeHAlign](#) properties

---

## TypeFloatDecimalChar Property

[See Also](#)

[Example](#)

### Description

Sets or returns the decimal point character used in a float cell. This property is available at run time only.

### Syntax

C++	<b>short</b> CSpreadSheet::GetTypeFloatDecimalChar( ); <b>void</b> CSpreadSheet::SetTypeFloatDecimalChar( <b>short</b> value);
Visual Basic	[form.]Spread.TypeFloatDecimalChar[ = value%]

### Remarks

Use the TypeFloatDecimalChar property with float cells.

You can use the ASCII value of the character to designate the decimal point character.

The default value for the TypeFloatDecimalChar property is 0. If this property is set to 0, the spreadsheet uses the default decimal point character designated by the [FloatDefDecimalChar](#) property. To override the FloatDefDecimalChar property value for an individual float cell, set the TypeFloatDecimalChar property to a value other than 0.

### Data Type

Integer

**See Also**

[CellType](#), [FloatDefDecimalChar](#), [TypeFloatCurrencyChar](#), [TypeFloatDecimalPlaces](#), [TypeFloatMax](#), [TypeFloatMin](#), [TypeFloatMoney](#), [TypeFloatSeparator](#), [TypeFloatSepChar](#), [TypeHAlign](#) properties

---

## TypeFloatDecimalPlaces Property

[See Also](#)

[Example](#)

### Description

Sets or returns the number of digits allowed to the right of the decimal point in a float cell. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetTypeFloatDecimalPlaces( );  
                         **void** CSpreadSheet::SetTypeFloatDecimalPlaces(**short** value);

Visual Basic        [form.]Spread.TypeFloatDecimalPlaces[ = value%]

### Remarks

Use the TypeFloatDecimalPlaces property with float cells.

The default value for the TypeFloatDecimalPlaces property is two decimal places.

### Data Type

Integer

**See Also**

[CellType](#), [TypeFloatCurrencyChar](#), [TypeFloatDecimalChar](#), [TypeFloatMax](#), [TypeFloatMin](#), [TypeFloatMoney](#), [TypeFloatSeparator](#), [TypeFloatSepChar](#), [TypeHAlign](#) properties



## TypeFloatMax Property

[See Also](#)

[Example](#)

### Description

Sets or returns the maximum value allowed in a float cell. This property is available at run time only.

### Syntax

C++ (OCX)	<b>double</b> CSpreadSheet::GetTypeFloatMax( ); <b>void</b> CSpreadSheet::SetTypeFloatMax( <b>double</b> value);
C++ (MFC VBX)	<b>CString</b> CSpreadSheet::GetTypeFloatMax( ); <b>void</b> CSpreadSheet::SetTypeFloatMax( <b>LPCSTR</b> value);
C++ (OWL VBX)	<b>string</b> CSpreadSheet::GetTypeFloatMax( ); <b>void</b> CSpreadSheet::SetTypeFloatMax( <b>LPCSTR</b> value);
Visual Basic	[form.]Spread.TypeFloatMax[ = text\$]

### Remarks

Use the TypeFloatMax property with float cells.

The default value for the TypeFloatMax property is 9999999.99.

### Data Type

VBX: String

OCX: Double

**See Also**

[CellType](#), [TypeFloatCurrencyChar](#), [TypeFloatDecimalChar](#), [TypeFloatDecimalPlaces](#), [TypeFloatMin](#), [TypeFloatMoney](#), [TypeFloatSeparator](#), [TypeFloatSepChar](#), [TypeHAlign](#) properties

## TypeFloatMin Property

[See Also](#)

[Example](#)

### Description

Sets or returns the minimum value allowed in a float cell. This property is available at run time only.

### Syntax

C++ (OCX)	<b>double</b> CSpreadSheet::GetTypeFloatMin( ); <b>void</b> CSpreadSheet::SetTypeFloatMin( <b>double</b> value);
C++ (MFC VBX)	<b>CString</b> CSpreadSheet::GetTypeFloatMin( ); <b>void</b> CSpreadSheet::SetTypeFloatMin( <b>LPCSTR</b> value);
C++ (OWL VBX)	<b>string</b> CSpreadSheet::GetTypeFloatMin( ); <b>void</b> CSpreadSheet::SetTypeFloatMin( <b>LPCSTR</b> value);
Visual Basic	[form.]Spread.TypeFloatMin[ = text\$]

### Remarks

Use the TypeFloatMin property with float cells.

The default value for the TypeFloatMin property is —9999999.99.

### Data Type

VBX: String  
OCX: Double

**See Also**

[CellType](#), [TypeFloatCurrencyChar](#), [TypeFloatDecimalChar](#), [TypeFloatDecimalPlaces](#), [TypeFloatMax](#), [TypeFloatMoney](#), [TypeFloatSeparator](#), [TypeFloatSepChar](#), [TypeHAlign](#) properties

## TypeFloatMoney Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the spreadsheet displays a currency symbol to the left of the value in a float cell. This property is available at run time only.

### Syntax

C++ **BOOL** CSpreadSheet::GetTypeFloatMoney( );  
**void** CSpreadSheet::SetTypeFloatMoney(**BOOL** value);

Visual Basic [form.]Spread.TypeFloatMoney[ = boolean%]

### Remarks

Use the TypeFloatMoney property with float cells.

The default value for the TypeFloatMoney property is False, which means no currency symbol is displayed.

If the TypeFloatMoney property is set to True, the currency symbol designated by the [TypeFloatCurrencyChar](#) property is used. If the TypeFloatCurrencyChar property is not set, the default currency symbol designated by the [FloatDefCurrencyChar](#) property is used. If neither of these properties is set, the currency symbol is taken from the international Currency Format setting in the Windows 3.1 or Windows NT Control Panel, or the regional Currency Symbol setting in the Windows 95 Control Panel.

**Note** The spreadsheet does not recognize changes made to the international settings in the Windows Control Panel until you restart Visual Basic or your application (or perform any operation that unloads and reloads the current .VBX or .OCX file).

### Data Type

Integer (Boolean)

**See Also**

[CellType](#), [FloatDefCurrencyChar](#), [TypeFloatCurrencyChar](#), [TypeFloatDecimalChar](#), [TypeFloatDecimalPlaces](#), [TypeFloatMax](#), [TypeFloatMin](#), [TypeFloatSeparator](#), [TypeFloatSepChar](#), [TypeHAlign](#) properties

## TypeFloatSeparator Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the thousands separator, such as the comma in "1,000", is displayed in a float cell. This property is available at run time only.

### Syntax

C++ **BOOL** CSpreadSheet::GetTypeFloatSeparator( );  
**void** CSpreadSheet::SetTypeFloatSeparator(**BOOL** value);

Visual Basic [form.]Spread.TypeFloatSeparator[ = *boolean%*]

### Remarks

Use the TypeFloatSeparator property with float cells.

The default value for the TypeFloatSeparator property is False.

If the TypeFloatSeparator property is set to True, the character displayed is taken from the [TypeFloatSepChar](#) property setting.

**Note** The spreadsheet does not recognize changes made to the international settings in the Windows Control Panel until you restart Visual Basic or your application (or perform any operation that unloads and reloads the current .VBX or .OCX file).

### Data Type

Integer (Boolean)

**See Also**

[CellType](#), [TypeFloatCurrencyChar](#), [TypeFloatDecimalChar](#), [TypeFloatDecimalPlaces](#), [TypeFloatMax](#), [TypeFloatMin](#), [TypeFloatMoney](#), [TypeFloatSepChar](#), [TypeHAlign](#) properties



## TypeFloatSepChar Property

[See Also](#)

[Example](#)

### Description

Sets or returns the character used to separate the thousands place from the hundreds place in a float cell. This property is available at run time only.

### Syntax

C++	<b>short</b> CSpreadSheet::GetTypeFloatSepChar( ); <b>void</b> CSpreadSheet::SetTypeFloatSepChar( <b>short</b> value);
Visual Basic	[form.]Spread.TypeFloatSepChar[ = value%]

### Remarks

Use the TypeFloatSepChar property with float cells.

The TypeFloatSeparator property must be set to True for the separator character to be shown.

You can use the ASCII value of the character to designate the separator character.

The default value for the TypeFloatSepChar property is 0. If this property is set to 0, the spreadsheet uses the default separator character designated by the [FloatDefSepChar](#) property. To override the FloatDefSepChar property value for an individual float cell, set the TypeFloatSepChar property to a value other than 0.

### Data Type

Integer

**See Also**

[CellType](#), [FloatDefSepChar](#), [TypeFloatCurrencyChar](#), [TypeFloatDecimalChar](#), [TypeFloatDecimalPlaces](#), [TypeFloatMax](#), [TypeFloatMin](#), [TypeFloatMoney](#), [TypeFloatSeparator](#), [TypeHAlign](#) properties

## TypeHAlign Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the text in a cell is centered, left-aligned, or right-aligned. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetTypeHAlign( );  
                         **void** CSpreadSheet::SetTypeHAlign(**short** value);

Visual Basic        [form.]Spread.TypeHAlign[ = setting%]

### Remarks

The following settings are available:

Setting	Description	Constant
0 - Left	Left aligns text	SS_CELL_H_ALIGN_LEFT
1 - Right	Right aligns text	SS_CELL_H_ALIGN_RIGHT
2 - Center	Centers text	SS_CELL_H_ALIGN_CENTER

### Data Type

Integer (Enumerated)

**See Also**

[CellType](#), [TypeTextAlignVert](#) properties

---

## TypeIntegerMax Property

[See Also](#)

[Example](#)

### Description

Sets or returns the maximum value allowed in an integer cell. This property is available at run time only.

### Syntax

C++	<b>long</b> CSpreadSheet::GetTypeIntegerMax( ); <b>void</b> CSpreadSheet::SetTypeIntegerMax( <b>long</b> value);
Visual Basic	[form.]Spread.TypeIntegerMax[ = value&]

### Remarks

Use the TypeIntegerMax property with integer cells.

The default value for the TypeIntegerMax property is 9,999,999.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets the `TypeIntegerMax` property to the default value.

C++

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Define cell as type INTEGER
vaSpread1->SetCellType(SS_CELL_TYPE_INTEGER);
vaSpread1->SetTypeIntegerMin(-99999999);
vaSpread1->SetTypeIntegerMax(99999999);
vaSpread1->SetTypeHAlign(SS_CELL_H_ALIGN_LEFT);
vaSpread1->SetTypeSpin(FALSE);
vaSpread1->SetTypeIntegerSpinInc(1);
vaSpread1->SetTypeIntegerSpinWrap(FALSE);
```

Visual Basic

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Define cell as type INTEGER
vaSpread1.CellType = SS_CELL_TYPE_INTEGER
vaSpread1.TypeIntegerMin = -99999999
vaSpread1.TypeIntegerMax = 99999999
vaSpread1.TypeHAlign = SS_CELL_H_ALIGN_LEFT
vaSpread1.TypeSpin = False
vaSpread1.TypeIntegerSpinInc = 1
vaSpread1.TypeIntegerSpinWrap = False
```

**See Also**

[CellType](#), [TypeHAlign](#), [TypeIntegerMin](#), [TypeIntegerSpinInc](#), [TypeIntegerSpinWrap](#), [TypeSpin](#) properties

---

## TypeIntegerMin Property

[See Also](#)

[Example](#)

### Description

Sets or returns the minimum value allowed in an integer cell. This property is available at run time only.

### Syntax

C++	<b>long</b> CSpreadSheet::GetTypeIntegerMin( ); <b>void</b> CSpreadSheet::SetTypeIntegerMin( <b>long</b> value);
Visual Basic	[form.]Spread.TypeIntegerMin[ = value&]

### Remarks

Use the TypeIntegerMin property with integer cells.

The default value for the TypeIntegerMin property is —9,999,999.

### Data Type

Long Integer



**See Also**

[CellType](#), [TypeHAlign](#), [TypeIntegerMax](#), [TypeIntegerSpinInc](#), [TypeIntegerSpinWrap](#), [TypeSpin](#) properties

---

## TypeIntegerSpinInc Property

[See Also](#)

[Example](#)

### Description

Sets or returns the increment used in an integer cell when the spin button is pressed. This property is available at run time only.

### Syntax

C++	<b>long</b> CSpreadSheet::GetTypeIntegerSpinInc( ); <b>void</b> CSpreadSheet::SetTypeIntegerSpinInc( <b>long</b> value);
Visual Basic	[form.]Spread.TypeIntegerSpinInc[ = value&]

### Remarks

Use the TypeIntegerSpinInc property with integer cells.

The default value for the TypeIntegerSpinInc property is 1.

### Data Type

Long Integer

**See Also**

[CellType](#), [TypeHAlign](#), [TypeIntegerMax](#), [TypeIntegerMin](#), [TypeIntegerSpinWrap](#), [TypeSpin](#) properties

---

## TypeIntegerSpinWrap Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the value of the spin button in an integer cell wraps when the minimum or maximum values are reached. This property is available at run time only.

### Syntax

C++ **BOOL** CSpreadSheet::GetTypeIntegerSpinWrap( );  
**void** CSpreadSheet::SetTypeIntegerSpinWrap(**BOOL** value);

Visual Basic [form.]Spread.TypeIntegerSpinWrap[ = boolean%]

### Remarks

Use the TypeIntegerSpinWrap property with integer cells.

The default value for the TypeIntegerSpinWrap property is False.

### Data Type

Integer (Boolean)

**See Also**

[CellType](#), [TypeHAlign](#), [TypeIntegerMax](#), [TypeIntegerMin](#), [TypeIntegerSpinInc](#), [TypeSpin](#) properties

---

## TypeOwnerDrawStyle Property

[See Also](#)

### Description

Sets or returns the style assigned to an owner-drawn cell. This property is available at run time only.

### Syntax

C++	<b>long</b> CSpreadSheet::GetTypeOwnerDrawStyle( ); <b>void</b> CSpreadSheet::SetTypeOwnerDrawStyle( <b>long</b> value);
Visual Basic	[form.]Spread.TypeOwnerDrawStyle[ = value&]

### Remarks

The style assigned to an owner-drawn cell with the TypeOwnerDrawStyle property is user-defined and is not used by the spreadsheet. It is sent to the application with the [DrawItem](#) event.

### Data Type

Long Integer

**See Also**

[CellType](#) property

[DrawItem](#) event

---

## TypePicDefaultText Property

[See Also](#)

[Example](#)

### Description

Sets or returns the default text displayed in a formatted PIC cell. This property is available at run time only.

### Syntax

C++ (OCX)	<b>CString</b> CSpreadSheet::GetTypePicDefaultText( ); <b>void</b> CSpreadSheet::SetTypePicDefaultText( <b>LPCWSTR</b> value);
C++ (MFC VBX)	<b>CString</b> CSpreadSheet::GetTypePicDefaultText( ); <b>void</b> CSpreadSheet::SetTypePicDefaultText( <b>LPCSTR</b> value);
C++ (OWL VBX)	<b>string</b> CSpreadSheet::GetTypePicDefaultText( ); <b>void</b> CSpreadSheet::SetTypePicDefaultText( <b>LPCSTR</b> value);
Visual Basic	[form.]Spread.TypePicDefaultText[ = text\$]

### Remarks

Use the TypePicDefaultText property with formatted PIC cells.

The default text is initially displayed in the cell. When the user deletes other characters within the cell, the deleted characters are replaced by the default text.

### Data Type

String



[Print](#)

[Copy](#)

[Close](#)

The following example specifies an empty string as the default text for a formatted PIC cell.

C++

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Define cell as type PIC
vaSpread1->SetCellType(SS_CELL_TYPE_PIC);
vaSpread1->SetTypePicDefaultText("");
vaSpread1->SetTypePicMask("(999)-UUU");
vaSpread1->SetTypeHAlign(SS_CELL_H_ALIGN_LEFT);
```

Visual Basic

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Define cell as type PIC
vaSpread1.CellType = SS_CELL_TYPE_PIC
vaSpread1.TypePicDefaultText = ""
vaSpread1.TypePicMask = "(999)-UUU"
vaSpread1.TypeHAlign = SS_CELL_H_ALIGN_LEFT
```

**See Also**

[CellType](#), [TypeHAlign](#), [TypePicMask](#) properties

## TypePicMask Property

[See Also](#)

[Example](#)

### Description

Sets or returns the mask used in a formatted PIC cell. This property is available at run time only.

### Syntax

```
C++ (OCX)      CString CSpreadSheet::GetTypePicMask( );  
               void CSpreadSheet::SetTypePicMask(LPCWSTR value);  
  
C++ (MFC VBX) CString CSpreadSheet::GetTypePicMask( );  
               void CSpreadSheet::SetTypePicMask(LPCSTR value);  
  
C++ (OWL VBX) string CSpreadSheet::GetTypePicMask( );  
               void CSpreadSheet::SetTypePicMask(LPCSTR value);  
  
Visual Basic  [form.]Spread.TypePicMask[ = text$]
```

### Remarks

Use the TypePicMask property with formatted PIC cells.

Each character in the mask represents one displayed character, with the exception of placeholders, which are prefixed by a slash character (/).

When the user types data into the cell, placeholder characters are automatically skipped. No entry is permitted in these positions, nor is the user able to delete them. If the slash character is followed by one of the listed formatting characters, that character is displayed as a placeholder.

Use the following characters to specify what type of data the user can type into the mask:

<b>Mask Character</b>	<b>Characters Allowed</b>
X	Any ASCII character
9	0—9, decimal
A	A—Z, a—z
N	A—Z, a—z, 0—9, decimal
U	A—Z
L	a—z
H	(Hexadecimal) 0—9, a—f, A—F

To specify one of the above characters as a placeholder, type a slash character (/) and then the character.

### Data Type

String

**See Also**

[CellType](#), [TypeHAlign](#), [TypePicDefaultText](#) properties

---

## TypePictCenter Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the picture in a picture cell is centered. This property is available at run time only.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetTypePictCenter( ); <b>void</b> CSpreadSheet::SetTypePictCenter( <b>BOOL</b> value);
Visual Basic	[form.]Spread.TypePictCenter[ = boolean%]

### Remarks

Use the TypePictCenter property with picture cells.

The default value for the TypePictCenter property is False. To center a picture in a picture cell, set this property to True.

### Data Type

Integer (Boolean)

[Print](#)

[Copy](#)

[Close](#)

The following example centers the picture in a picture cell.

#### C++ (VBX)

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Define cell as type PICTURE
vaSpread1->SetCellType(SS_CELL_TYPE_PICTURE);
vaSpread1->SetTypePictCenter(TRUE);
vaSpread1->SetTypePictMaintainScale(TRUE);
vaSpread1->SetTypePictStretch(TRUE);
// Convert hBitmap to hPic for the VBX using C++
PIC picture1;
HPIC hPic1;
HBITMAP hBitmap1;
hBitmap1=LoadBitmap(AfxGetInstanceHandle( ),"pict1");
picture1.picData.bmp.hbitmap1=hBitmap1;
picture1.picType=PICTURE_BITMAP;
hPic1=AfxSetPict(NULL,&picture1);
vaSpread1->SetTypePictPicture(hPic1);
```

#### Visual Basic

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Define cell as type PICTURE
vaSpread1.CellType = SS_CELL_TYPE_PICTURE
vaSpread1.TypePictCenter = True
vaSpread1.TypePictMaintainScale = True
vaSpread1.TypePictStretch = True
vaSpread1.TypePictPicture = LoadPicture("c:\pict1.bmp")
```

**See Also**

[CellType](#), [TypePictMaintainScale](#), [TypePictPicture](#), [TypePictStretch](#) properties

---

## TypePictMaintainScale Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the application forces a picture in a picture cell to proportionally maintain its width and height when stretched. This property is available at run time only.

### Syntax

C++ **BOOL** CSpreadSheet::GetTypePictMaintainScale( );  
**void** CSpreadSheet::SetTypePictMaintainScale(**BOOL** value);

Visual Basic [form.]Spread.TypePictMaintainScale[ = boolean%]

### Remarks

Use the TypePictMaintainScale property with picture cells.

The default value for the TypePictMaintainScale property is False.

Use the TypePictMaintainScale property in conjunction with the [TypePictStretch](#) property. When the TypePictStretch property is set to True, the picture will stretch to fit into the cell regardless of the proportions of the picture unless the TypePictMaintainScale property is set to True, which sizes the picture to fit at the maximum size it can without changing its proportions.

### Data Type

Integer (Boolean)



**See Also**

[CellType](#), [TypePictCenter](#), [TypePictPicture](#), [TypePictStretch](#) properties

## TypePictPicture Property

[See Also](#)

[Example](#)

### Description

Sets or returns the picture displayed in a picture cell. This property is available at run time only.

### Syntax

C++ (OCX)	<b>CPicture</b> CSpreadSheet::GetTypePictPicture( ); <b>void</b> CSpreadSheet::SetTypePictPicture( <b>LPDISPATCH</b> value);
C++ (VBX)	<b>HPIC</b> CSpreadSheet::GetTypePictPicture( ); <b>void</b> CSpreadSheet::SetTypePictPicture( <b>HPIC</b> value);
Visual Basic	[form.]Spread.TypePictPicture[ = picture]

### Remarks

Use the TypePictPicture property with picture cells.

You can display either bitmaps or icons in picture cells.

### Data Type

Picture

**See Also**

[CellType](#), [TypePictCenter](#), [TypePictMaintainScale](#), [TypePictStretch](#) properties

---

## TypePictStretch Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether a picture in a picture cell is stretched to fit the size of the cell. This property is available at run time only.

### Syntax

C++                    **BOOL** CSpreadSheet::GetTypePictStretch( );  
                         **void** CSpreadSheet::SetTypePictStretch(**BOOL** value);

Visual Basic        [form.]Spread.TypePictStretch[ = *boolean%*]

### Remarks

Use the TypePictStretch property with picture cells.

The default value for the TypePictStretch property is False.

When the TypePictStretch property is set to True, the picture stretches to fit into the cell regardless of the original proportions of the picture. By setting the [TypePictMaintainScale](#) property to True, the picture maintains its proportions while stretching to its maximum size within the cell.

### Data Type

Integer (Boolean)

**See Also**

[CellType](#), [TypePictCenter](#), [TypePictMaintainScale](#), [TypePictPicture](#) properties

---

## TypeSpin Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether a spin button is displayed in a date, time, or integer cell. This property is available at run time only.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetTypeSpin( ); <b>void</b> CSpreadSheet::SetTypeSpin( <b>BOOL</b> value);
Visual Basic	[form.]Spread.TypeSpin[ = boolean%]

### Remarks

The default value for the TypeSpin property is False.

### Data Type

Integer (Boolean)

**See Also**

[CellType](#), [TypeIntegerMax](#), [TypeIntegerMin](#), [TypeIntegerSpinWrap](#) properties

## TypeTextAlignVert

[See Also](#)   [Example](#)

### Description

Sets or returns the vertical alignment of the text in a static text cell. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetTypeTextAlignVert( );  
                         **void** CSpreadSheet::SetTypeTextAlignVert(**short** value);

Visual Basic        [form.]Spread.TypeTextAlignVert[ = setting%]

### Remarks

Use the TypeTextAlignVert property with static text cells.

The following settings are available:

Setting	Description	Constant
0 - Bottom	Bottom aligns text	SS_CELL_STATIC_V_ALIGN_BOTTOM
1 - Center	Centers text	SS_CELL_STATIC_V_ALIGN_CENTER
2 - Top	(Default) Top aligns text	SS_CELL_STATIC_V_ALIGN_TOP

### Data Type

Integer (Enumerated)



[Print](#)

[Copy](#)

[Close](#)

The following example top aligns the text in a static text cell.

C++

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Define cell as type STATIC
vaSpread1->SetCellType(SS_CELL_TYPE_STATIC_TEXT);
vaSpread1->SetTypeHAlign(SS_CELL_H_ALIGN_LEFT);
vaSpread1->SetTypeTextAlignVert(SS_CELL_STATIC_V_ALIGN_TOP);
vaSpread1->SetTypeTextShadow(FALSE);
vaSpread1->SetTypeTextWordWrap(FALSE);
vaSpread1->SetTypeTextPrefix(FALSE);
vaSpread1->SetTypeTextShadowIn(FALSE);
```

Visual Basic

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Define cell as type STATIC
vaSpread1.CellType = SS_CELL_TYPE_STATIC_TEXT
vaSpread1.TypeHAlign = SS_CELL_H_ALIGN_LEFT
vaSpread1.TypeTextAlignVert = SS_CELL_STATIC_V_ALIGN_TOP
vaSpread1.TypeTextShadow = False
vaSpread1.TypeTextWordWrap = False
vaSpread1.TypeTextPrefix = False
vaSpread1.TypeTextShadowIn = False
```

**See Also**

[CellType](#), [TypeHAlign](#), [TypeTextPrefix](#), [TypeTextShadow](#), [TypeTextShadowIn](#), [TypeTextWordWrap](#) properties

---

## TypeTextPrefix Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether typing an ampersand (&) in text in a static text cell underlines the next character. This property is available at run time only.

### Syntax

C++ **BOOL** CSpreadSheet::GetTypeTextPrefix( );  
**void** CSpreadSheet::SetTypeTextPrefix(**BOOL** value);

Visual Basic [form.]Spread.TypeTextPrefix[ = boolean%]

### Remarks

Use the TypeTextPrefix property with static text cells.

The default value for the TypeTextPrefix property is False.

When this property is set to True, typing an ampersand (&) before a character specifies to underline that character when it is displayed.

### Data Type

Integer (Boolean)

**See Also**

[CellType](#), [TypeHAlign](#), [TypeTextAlignVert](#), [TypeTextShadow](#), [TypeTextShadowIn](#), [TypeTextWordWrap](#) properties

---

## TypeTextShadow Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether text in a static text cell is displayed with a three-dimensional shadow. This property is available at run time only.

### Syntax

C++ **BOOL** CSpreadSheet::GetTypeTextShadow( );  
**void** CSpreadSheet::SetTypeTextShadow(**BOOL** value);

Visual Basic [form.]Spread.TypeTextShadow[ = boolean%]

### Remarks

Use the TypeTextShadow property with static text cells.

The default value for the TypeTextShadow property is True for headers and False for all other static text cells.

You can also use the TypeTextShadow property on the column and row headers by specifying the header column or row and setting the value to True.

### Data Type

Integer (Boolean)

**See Also**

[CellType](#), [TypeHAlign](#), [TypeTextAlignVert](#), [TypeTextPrefix](#), [TypeTextShadowIn](#), [TypeTextWordWrap](#) properties

---

## TypeTextShadowIn Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether a static text cell is displayed with a shadow effect projecting inside the cell. This property is available at run time only.

### Syntax

C++ **BOOL** CSpreadSheet::GetTypeTextShadowIn( );  
**void** CSpreadSheet::SetTypeTextShadowIn(**BOOL** value);

Visual Basic [form.]Spread.TypeTextShadowIn[ = boolean%]

### Remarks

Use the TypeTextShadowIn property with static text cells.

The default value for the TypeTextShadowIn property is False.

The TypeTextShadowIn property has no effect unless the [TypeTextShadow](#) property is set to True.

### Data Type

Integer (Boolean)

**See Also**

[CellType](#), [TypeHAlign](#), [TypeTextAlignVert](#), [TypeTextPrefix](#), [TypeTextShadow](#), [TypeTextWordWrap](#) properties



---

## TypeTextWordWrap Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether wordwrapping is allowed in a static text cell. This property is available at run time only.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetTypeTextWordWrap( ); <b>void</b> CSpreadSheet::SetTypeTextWordWrap( <b>BOOL</b> value);
Visual Basic	[form.]Spread.TypeTextWordWrap[ = boolean%]

### Remarks

Use the TypeTextWordWrap property with static text cells.

The default value for the TypeTextWordWrap property is False *except* for cells in the column header row and the row header column. The default value for the TypeTextWordWrap property for header cells is True.

When the TypeTextWordWrap property is set to True, text is wrapped when it is wider than the width of the cell. When the TypeTextWordWrap property is set to False, text that is wider than the cell is truncated at the edge of the cell. Use a linefeed character (ASCII 10 or '\n' in C++) to force a line break.

### Data Type

Integer (Boolean)

**See Also**

[CellType](#), [TypeHAlign](#), [TypeTextAlignVert](#), [TypeTextPrefix](#), [TypeTextShadow](#), [TypeTextShadowIn](#) properties

## TypeTime24Hour Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether time is displayed in 12- or 24-hour format in a time cell. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetTypeTime24Hour( );  
                         **void** CSpreadSheet::SetTypeTime24Hour(**short** value);

Visual Basic        [form.]Spread.TypeTime24Hour[ = setting%]

### Remarks

Use the TypeTime24Hour property with time cells.

The default value for the TypeTime24Hour property is taken from the Time Format setting in the Windows Control Panel.

**Note** The spreadsheet does not recognize changes made to the settings in the Windows Control Panel until you restart Visual Basic or your application (or perform any operation that unloads and reloads the current .VBX or .OCX file).

The following settings are available:

Setting	Description	Constant
0 - 12 Hour	Displays time in 12-hour format	SS_CELL_TIME_12_HOUR_CLOCK
1 - 24 Hour	Displays time in 24-hour format	SS_CELL_TIME_24_HOUR_CLOCK

### Data Type

Integer (Enumerated)

[Print](#)

[Copy](#)

[Close](#)

The following example specifies that time values in a time cell are displayed in 12-hour format.

C++

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Define cell as type TIME
vaSpread1->SetCellType(SS_CELL_TYPE_TIME);
vaSpread1->SetTypeTime24Hour(SS_CELL_TIME_12_HOUR_CLOCK);
vaSpread1->SetTypeTimeSeconds(FALSE);
vaSpread1->SetTypeSpin(FALSE);
vaSpread1->SetTypeHAlign(SS_CELL_H_ALIGN_LEFT);
vaSpread1->SetTypeTimeSeparator(58);
vaSpread1->SetTypeTimeMin("0");
vaSpread1->SetTypeTimeMax("235959");
```

Visual Basic

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Define cell as type TIME
vaSpread1.CellType = SS_CELL_TYPE_TIME
vaSpread1.TypeTime24Hour = SS_CELL_TIME_12_HOUR_CLOCK
vaSpread1.TypeTimeSeconds = False
vaSpread1.TypeSpin = False
vaSpread1.TypeHAlign = SS_CELL_H_ALIGN_LEFT
vaSpread1.TypeTimeSeparator = 58
vaSpread1.TypeTimeMin = "0"
vaSpread1.TypeTimeMax = "235959"
```

**See Also**

[CellType](#), [TypeHAlign](#), [TypeSpin](#), [TypeTimeMax](#), [TypeTimeMin](#), [TypeTimeSeconds](#), [TypeTimeSeparator](#) properties

## TypeTimeMax Property

[See Also](#)

[Example](#)

### Description

Sets or returns the maximum time value allowed in a time cell. This property is available at run time only.

### Syntax

C++ (OCX)	<b>CString</b> CSpreadSheet::GetTypeTimeMax( ); <b>void</b> CSpreadSheet::SetTypeTimeMax( <b>LPCSTR</b> value);
C++ (MFC VBX)	<b>CString</b> CSpreadSheet::GetTypeTimeMax( ); <b>void</b> CSpreadSheet::SetTypeTimeMax( <b>LPCSTR</b> value);
C++ (OWL VBX)	<b>string</b> CSpreadSheet::GetTypeTimeMax( ); <b>void</b> CSpreadSheet::SetTypeTimeMax( <b>LPCSTR</b> value);
Visual Basic	[form.]Spread.TypeTimeMax[ = text\$]

### Remarks

Use the TypeTimeMax property with time cells.

Use the format "HHMMSS" for setting the maximum time, where the time is in 24-hour format.

The default value for the TypeTimeMax property is "235959".

### Data Type

String

**See Also**

[CellType](#), [TypeHAlign](#), [TypeSpin](#), [TypeTime24Hour](#), [TypeTimeMin](#), [TypeTimeSeconds](#), [TypeTimeSeparator](#) properties

## TypeTimeMin Property

[See Also](#)

[Example](#)

### Description

Sets or returns the minimum time value allowed in a time cell. This property is available at run time only.

### Syntax

C++ (OCX)	<b>CString</b> CSpreadSheet::GetTypeTimeMin( ); <b>void</b> CSpreadSheet::SetTypeTimeMin( <b>LPCTSTR</b> value);
C++ (MFC VBX)	<b>CString</b> CSpreadSheet::GetTypeTimeMin( ); <b>void</b> CSpreadSheet::SetTypeTimeMin( <b>LPCSTR</b> value);
C++ (OWL VBX)	<b>string</b> CSpreadSheet::GetTypeTimeMin( ); <b>void</b> CSpreadSheet::SetTypeTimeMin( <b>LPCSTR</b> value);
Visual Basic	[form.]Spread.TypeTimeMin[ = text\$]

### Remarks

Use the TypeTimeMin property with time cells.

Use the format "HHMMSS" for setting the minimum time, where the time is in 24-hour format.

The default value for the TypeTimeMin property is "000000".

### Data Type

String



**See Also**

[CellType](#), [TypeHAlign](#), [TypeSpin](#), [TypeTime24Hour](#), [TypeTimeMax](#), [TypeTimeSeconds](#), [TypeTimeSeparator](#) properties

## TypeTimeSeconds Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether seconds are displayed in a time cell. This property is available at run time only.

### Syntax

C++ (OCX)	<b>BOOL</b> CSpreadSheet::GetTypeTimeSeconds( ); <b>void</b> CSpreadSheet::SetTypeTimeSeconds( <b>BOOL</b> value);
C++ (VBX)	<b>short</b> CSpreadSheet::GetTypeTimeSeconds( ); <b>void</b> CSpreadSheet::SetTypeTimeSeconds( <b>short</b> value);
Visual Basic	[form.]Spread.TypeTimeSeconds[ = boolean%]

### Remarks

Use the TypeTimeSeconds property with time cells.

The default value for the TypeTimeSeconds property is False.

When the TypeTimeSeconds property is set to True, seconds are displayed in the time cell.

### Data Type

Integer

**See Also**

[CellType](#), [TypeHAlign](#), [TypeSpin](#), [TypeTime24Hour](#), [TypeTimeMax](#), [TypeTimeMin](#), [TypeTimeSeparator](#) properties

---

## TypeTimeSeparator Property

[See Also](#)

[Example](#)

### Description

Sets or returns the character used to separate the hours, minutes, and seconds in a time cell. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetTypeTimeSeparator( );  
                         **void** CSpreadSheet::SetTypeTimeSeparator(**short** value);

Visual Basic        [form.]Spread.TypeTimeSeparator[ = value%]

### Remarks

Use the TypeTimeSeparator property with time cells.

The default value for the TypeTimeSeparator property is taken from the Time Format setting in the Windows Control Panel.

**Note**    The spreadsheet does not recognize changes made to the settings in the Windows Control Panel until you restart Visual Basic or your application (or perform any operation that unloads and reloads the current .VBX or .OCX file).

Use the ASCII value of the character to specify the time separator character.

### Data Type

Integer

**See Also**

[CellType](#), [TypeHAlign](#), [TypeSpin](#), [TypeTime24Hour](#), [TypeTimeMax](#), [TypeTimeMin](#), [TypeTimeSeconds](#) properties

## UnitType Property

[See Also](#)

[Example](#)

### Description

Sets or returns the units used for the column width and row height.

### Syntax

C++ (OCX)      **long** CSpreadSheet::GetUnitType( );  
                  **void** CSpreadSheet::SetUnitType(**long** value);

C++ (VBX)      **short** CSpreadSheet::GetUnitType( );  
                  **void** CSpreadSheet::SetUnitType(**short** value);

Visual Basic    [form.]Spread.UnitType[ = setting%]

### OCX Use

Normal, VGA, or Twips option buttons under Unit Type on the [Op\\_Mode](#) property page

### Remarks

If you change the UnitType property, set it before any other property is referenced within the Form\_Load event. The application cannot change this value once you set it.

The following settings are available:

Setting	Description	Constant
0 - Normal	Column width and row height are based on system fixed font	SS_CELL_UNIT_NORMAL
1 - VGA Base	Column width and row height are based on predetermined value	SS_CELL_UNIT_VGA
2 - Twips	Units are in twips (1/1440 inch)	SS_CELL_UNIT_TWIPS

Settings 0 (Normal) and 1 (VGA Base) are very similar. You probably will not notice any difference between the settings unless your display is set to a high resolution.

### Data Type

Integer (Enumerated)

[Print](#)

[Copy](#)

[Close](#)

The following example sets the UnitType property to twips.

C++

```
// Set column width and row height unit type  
vaSpread1->SetUnitType(SS_CELL_UNIT_TWIPS);
```

Visual Basic

```
' Set column width and row height unit type  
vaSpread1.UnitType = SS_CELL_UNIT_TWIPS
```

**See Also**

[ColWidth](#), [RowHeight](#) properties



## UserResize Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the user can resize columns, rows, or both.

### Syntax

C++                    **short** CSpreadSheet::GetUserResize( );  
                         **void** CSpreadSheet::SetUserResize(**short** value);

Visual Basic        [form.]Spread.UserResize[ = value%]

### OCX Use

Columns and Rows check boxes under User Resize on the [Op Mode](#) property page

### Remarks

The following values are available. Note that you can combine values 1 (Cols) and 2 (Rows) using the Or operator. The default value is 3 (combination of values 1 and 2), which lets the user resize both columns and rows. Setting the UserResize property to 0 prevents the user from resizing both columns and rows.

Value	Description	Constant
1 - Cols	Lets the user resize columns	SS_USER_RESIZE_COL
2 - Rows	Lets the user resize rows	SS_USER_RESIZE_ROW

### Data Type

Integer

[Print](#)

[Copy](#)

[Close](#)

The following example lets the user resize both columns and rows.

C++

```
// Let user resize both columns and rows  
vaSpread1->UserResize(SS_USER_RESIZE_COL | SS_USER_RESIZE_ROW);
```

Visual Basic

```
' Let user resize both columns and rows  
vaSpread1.UserResize = SS_USER_RESIZE_COL or SS_USER_RESIZE_ROW
```

**See Also**

[UserResizeCol](#), [UserResizeRow](#) properties  
[ColWidthChange](#), [RowHeightChange](#) events

## UserResizeCol Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the user can resize individual columns. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetUserResizeCol( );  
                         **void** CSpreadSheet::SetUserResizeCol(**short** value);

Visual Basic        [form.]Spread.UserResizeCol[ = setting%]

### Remarks

The following settings are available:

Setting	Description	Constant
0 - Default	(Default) Uses <a href="#">UserResize</a> property setting	SS_USER_RESIZE_DEFAULT
1 - On	Lets user resize specified column	SS_USER_RESIZE_ON
2 - Off	Does not let user resize specified column	SS_USER_RESIZE_OFF

Set the [Col](#) property to select a column before using the UserResizeCol property.

### Data Type

Integer (Enumerated)

[Print](#)

[Copy](#)

[Close](#)

The following example uses the UserResize property setting to determine whether the user can resize a column and row.

C++

```
// Select a cell
vaSpread1->SetRow(2);
vaSpread1->SetCol(2);
// Use default resize setting
vaSpread1->SetUserResizeCol(SS_USER_RESIZE_DEFAULT);
vaSpread1->SetUserResizeRow(SS_USER_RESIZE_DEFAULT);
```

Visual Basic

```
' Select a cell
vaSpread1.Row = 2
vaSpread1.Col = 2
' Use default resize setting
vaSpread1.UserResizeCol = SS_USER_RESIZE_DEFAULT
vaSpread1.UserResizeRow = SS_USER_RESIZE_DEFAULT
```

**See Also**

[UserResize](#), [UserResizeRow](#) properties

[ColWidthChange](#) event

## UserResizeRow Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the user can resize individual rows. This property is available at run time only.

### Syntax

C++                    **short** CSpreadSheet::GetUserResizeRow( );  
                         **void** CSpreadSheet::SetUserResizeRow(**short value**);

Visual Basic        [form.]Spread.UserResizeRow[ = setting%]

### Remarks

The following settings are available:

Setting	Description	Constant
0 - Default	(Default) Uses <a href="#">UserResize</a> property setting	SS_USER_RESIZE_DEFAULT
1 - On	Lets user resize specified row	SS_USER_RESIZE_ON
2 - Off	Does not let user resize specified row	SS_USER_RESIZE_OFF

Set the [Row](#) property to select a row before using the UserResizeRow property.

### Data Type

Integer (Enumerated)

**See Also**

[UserResize](#), [UserResizeCol](#) properties

[RowHeightChange](#) event



## Value Property

[See Also](#)   [Example](#)

### Description

Sets or returns unformatted data. This property is available at run time only.

### Syntax

C++ (OCX)	<b>CString</b> CSpreadSheet::GetValue( ); <b>void</b> CSpreadSheet::SetValue( <b>LPCWSTR</b> value);
C++ (MFC VBX)	<b>CString</b> CSpreadSheet::GetValue( ); <b>void</b> CSpreadSheet::SetValue( <b>LPCSTR</b> value);
C++ (OWL VBX)	<b>string</b> CSpreadSheet::GetValue( ); <b>void</b> CSpreadSheet::SetValue( <b>LPCSTR</b> value);
Visual Basic	[form.]Spread.Value[ = text\$]

### Remarks

The following table describes how each cell type works with data assigned to or returned from the Value property.

Cell Type	Description
Date	Formats date value as "MMDDYYYY"
Edit	Performs validation (If the <a href="#">TypeEditMultiLine</a> property is set to True, use a linefeed character (ASCII 10 or '\n' in C++) to force a line break.)
Float	No formatting characters
Integer	No formatting characters
PIC	Strips the PIC value of all nonmask characters
Static Text	If the <a href="#">TypeTextWordWrap</a> property is set to True, use a linefeed character (ASCII 10 or '\n' in C++) to force a line break when setting text
Time	Formats time value as "HHMMSS"
Button	If the cell is a two-state button, sets or returns a 1 if the button is down or a 0 if the button is up
Combo Box	Index value of the currently selected item
Picture	Not used
Check Box	Sets or returns a 1 if the check box is checked or a 0 if the check box is not checked
Owner-Drawn	String only

### Data Type

String

[Print](#)

[Copy](#)

[Close](#)

The following example sets the Value property for a U.S. telephone number.

C++

```
vaSpread1->SetRow(2);  
vaSpread1->SetCol(2);  
vaSpread1->SetCellType(SS_CELL_TYPE_PIC);  
vaSpread1->SetTypePicMask("(999) 999-9999");  
vaSpread1->SetValue("2225551234");
```

Visual Basic

```
vaSpread1.Row = 2  
vaSpread1.Col = 2  
vaSpread1.CellType = SS_CELL_TYPE_PIC  
vaSpread1.TypePicMask = "(999) 999-9999"  
vaSpread1.Value = "2225551234"
```

**See Also**

[CellType](#), [Clip](#), [ClipValue](#), [Text](#) properties

---

## VirtualCurRowCount Property

[See Also](#)

[Example](#)

### Description

Returns the number of rows in the virtual buffer. This property is available at run time only.

### Syntax

C++ **long** CSpreadSheet::GetVirtualCurRowCount( );

Visual Basic [form.]Spread.VirtualCurRowCount

### Remarks

The VirtualCurRowCount property is valid only when virtual mode is on.

Use the [VirtualMode](#) property to turn virtual mode on or off.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example retrieves the number of rows in the virtual buffer.

C++

```
x = vaSpread1->GetVirtualCurRowCount ( );
```

Visual Basic

```
x = vaSpread1.VirtualCurRowCount
```

**See Also**

[VirtualCurTop](#), [VirtualMaxRows](#), [VirtualMode](#), [VirtualOverlap](#), [VirtualRows](#), [VirtualScrollBuffer](#) properties

---

## VirtualCurTop Property

[See Also](#)    [Example](#)

### Description

Returns the current top row of the virtual buffer. This property is available at run time only.

### Syntax

C++                    **long** CSpreadSheet::GetVirtualCurTop( );  
Visual Basic        [form.]Spread.VirtualCurTop

### Remarks

The VirtualCurTop property is valid only when virtual mode is on.

Use the [VirtualMode](#) property to turn virtual mode on or off.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example retrieves the current top row of the virtual buffer.

C++

```
x = vaSpread1->GetVirtualCurTop( );
```

Visual Basic

```
x = vaSpread1.VirtualCurTop
```



**See Also**

[VirtualCurRowCount](#), [VirtualMaxRows](#), [VirtualMode](#), [VirtualOverlap](#), [VirtualRows](#), [VirtualScrollBuffer](#) properties

## VirtualMaxRows Property

[See Also](#)

[Example](#)

### Description

Sets or returns the maximum number of rows allowed when virtual mode is on.

### Syntax

C++	<b>long</b> CSpreadSheet::GetVirtualMaxRows( ); <b>void</b> CSpreadSheet::SetVirtualMaxRows( <b>long</b> value);
Visual Basic	[form.]Spread.VirtualMaxRows[= value&]

### OCX Use

Virtual Max Rows box on the [Virtual Mode](#) property page

### Remarks

When virtual mode is on, you must use the VirtualMaxRows property instead of the [MaxRows](#) property to set the maximum number of rows. Use a value of — 1 if you do not know the exact number of rows. The default value for the VirtualMaxRows property is

— 1.

Use the [VirtualMode](#) property to turn virtual mode on or off.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets the maximum number of rows during virtual mode to 1000.

C++

```
// Set the number of rows or records in the data source
// Use -1 when the number of records is unknown
vaSpread1->SetVirtualMaxRows(1000);
// Set the max number of rows or records to keep in memory
vaSpread1->SetVirtualOverlap(100);
// Set the number of rows or records to read when additional
// data is requested
vaSpread1->SetVirtualRows(10);
// Enable virtual mode
vaSpread1->SetVirtualMode(TRUE);
// Reflect the number of rows in the virtual buffer and not
// the max number of rows or records
vaSpread1->SetVirtualScrollBuffer(FALSE);
// Enable special virtual scroll bars
vaSpread1->SetVScrollSpecial(TRUE);
```

Visual Basic

```
' Set the number of rows or records in the data source
' Use -1 when the number of records is unknown
vaSpread1.VirtualMaxRows = 1000
' Set the max number of rows or records to keep in memory
vaSpread1.VirtualOverlap = 100
' Set the number of rows or records to read when additional
' data is requested
vaSpread1.VirtualRows = 10
' Enable virtual mode
vaSpread1.VirtualMode = True
' Reflect the number of rows in the virtual buffer and not
' the max number of rows or records
vaSpread1.VirtualScrollBuffer = False
' Enable special virtual scroll bars
vaSpread1.VScrollSpecial = True
```

**See Also**

[VirtualCurRowCount](#), [VirtualCurTop](#), [VirtualMode](#), [VirtualOverlap](#), [VirtualRows](#), [VirtualScrollBuffer](#) properties

## VirtualMode Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether virtual mode is on or off.

### Syntax

```
C++          BOOL CSpreadSheet::GetVirtualMode( );  
             void CSpreadSheet::SetVirtualMode(BOOL value);  
  
Visual Basic [form.]Spread.VirtualMode[= boolean%]
```

### OCX Use

Virtual Mode check box on the [Virtual Mode](#) property page

### Remarks

The default value for the VirtualMode property is False.

Use virtual mode to inform the spreadsheet to store only a specified number of rows of data in memory at a time.

For more information on virtual mode, see [Using Virtual Mode](#).

### Notes

Keep in mind the following when using virtual mode:

- If you are using a bound spreadsheet that is using virtual mode, the Data control bound to the spreadsheet might not behave as you expect. For more information, see [Using Virtual Mode with Bound Spreadsheets](#).
- When printing a spreadsheet that is using virtual mode, set the [PrintUseDataMax](#) property to False.
- Do not use the [Action](#) property setting 32 (Smart Print) to print a spreadsheet that is using virtual mode. Use the Action property setting 13 (Print) instead.
- If the spreadsheet does not know the total number of rows, the row header column might not display the correct numbers for the rows. For best results, hide the row header column.
- For best results, do not use multiple- or extended-selection operation mode in a spreadsheet that is using virtual mode.
- Do not freeze rows in a spreadsheet that is using virtual mode.
- You cannot sort a spreadsheet that is using virtual mode.

### Data Type

Integer (Boolean)

**See Also**

[Action](#) (30 - VRefresh Buffer), [VirtualCurRowCount](#), [VirtualCurTop](#), [VirtualMaxRows](#), [VirtualOverlap](#), [VirtualRows](#), [VirtualScrollBuffer](#) properties

[QueryData](#) event

---

## VirtualOverlap Property

[See Also](#)

[Example](#)

### Description

Sets or returns the number of retrieved rows to keep in the virtual buffer when virtual mode is on.

### Syntax

C++	<b>long</b> CSpreadSheet::GetVirtualOverlap( ); <b>void</b> CSpreadSheet::SetVirtualOverlap( <b>long</b> value);
Visual Basic	[form.]Spread.VirtualOverlap[ = value&]

### OCX Use

Viewed Records to Keep in Memory box on the [Virtual Mode](#) property page

### Remarks

The default value for the VirtualOverlap property is False.

The overlap area is the number of previously retrieved rows kept in memory when a new buffer is read. The VirtualOverlap property specifies the maximum number of records to be retained in memory.

The VirtualOverlap property is valid only when virtual mode is on. Use the [VirtualMode](#) property to turn virtual mode on or off.

### Data Type

Long Integer

**See Also**

[VirtualCurRowCount](#), [VirtualCurTop](#), [VirtualMaxRows](#), [VirtualMode](#), [VirtualRows](#), [VirtualScrollBuffer](#) properties



---

## VirtualRows Property

[See Also](#)

[Example](#)

### Description

Sets or returns the number of virtual rows the spreadsheet requests at a time.

### Syntax

C++	<b>long</b> CSpreadSheet::GetVirtualRows( ); <b>void</b> CSpreadSheet::SetVirtualRows( <b>long</b> value);
Visual Basic	[form.]Spread.VirtualRows[ = value&]

### OCX Use

Rows Requested Each Time box on the [Virtual Mode](#) property page

### Remarks

The default value for the VirtualRows property is 0.

The VirtualRows property is valid only when virtual mode is on. The minimum number of virtual rows the spreadsheet can request is one screen of data. The spreadsheet sends the [QueryData](#) event when more data is needed.

Use the [VirtualMode](#) property to turn virtual mode on or off.

### Data Type

Long Integer

**See Also**

[VirtualCurRowCount](#), [VirtualCurTop](#), [VirtualMaxRows](#), [VirtualMode](#), [VirtualOverlap](#), [VirtualScrollBuffer](#) properties

[QueryData](#) event

---

## VirtualScrollBuffer Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the scroll box reflects the number of rows in the virtual buffer rather than the maximum number of rows.

### Syntax

C++                    **BOOL** CSpreadSheet::GetVirtualScrollBuffer( );  
                         **void** CSpreadSheet::SetVirtualScrollBuffer(**BOOL** value);

Visual Basic        [form.]Spread.VirtualScrollBuffer[ = boolean%]

### OCX Use

Box Reflects Rows in Buffer check box under Vertical Scroll Bar on the [Virtual Mode](#) property page

### Remarks

The default value for the VirtualScrollBuffer property is False.

The VirtualScrollBuffer property is valid only when virtual mode is on.

Use the [VirtualMode](#) property to turn virtual mode on or off.

### Data Type

Integer (Boolean)

**See Also**

[VirtualCurRowCount](#), [VirtualCurTop](#), [VirtualMaxRows](#), [VirtualMode](#), [VirtualOverlap](#), [VirtualRows](#) properties

---

## Visible Property

### Description

Sets or returns whether the Spread control is visible.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetVisible( ); <b>void</b> CSpreadSheet::SetVisible( <b>BOOL</b> value);
Visual Basic	[form.]Spread.Visible[ = boolean%]

### OCX Use

The Visible property is a standard extender property that can be provided by your container.

### Remarks

The default value for the Visible property is True.

When the Visible property is set to False, the Spread control is hidden.

To hide a control at startup, set the Visible property to False at design time. Setting this property in code lets you hide the control and later display it at run time in response to a particular event.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

Integer (Boolean)

---

## VisibleCols Property

[See Also](#)

[Example](#)

### Description

Sets or returns the number of columns that are fully displayed.

### Syntax

C++	<b>long</b> CSpreadSheet::GetVisibleCols( ); <b>void</b> CSpreadSheet::SetVisibleCols( <b>long</b> value);
Visual Basic	[form.]Spread.VisibleCols[ = value&]

### OCX Use

Number of Visible Columns box under Col Headers on the [Headers](#) property page

### Remarks

The default value for the VisibleCols property is 0.

Use the VisibleCols property when the [AutoSize](#) property is set to True. If the VisibleCols property is set to 0, the application uses the [MaxCols](#) property value.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets the number of visible columns to 3.

**C++**

```
// Set the number of visible columns  
vaSpread1->SetVisibleCols(3);  
vaSpread1->SetAutoSize(TRUE);
```

**Visual Basic**

```
' Set the number of visible columns  
vaSpread1.VisibleCols = 3  
vaSpread1.AutoSize = True
```

**See Also**

[AutoSize](#), [MaxCols](#), [VisibleRows](#) properties



---

## VisibleRows Property

[See Also](#)

[Example](#)

### Description

Sets or returns the number of rows that are fully displayed.

### Syntax

C++	<b>long</b> CSpreadSheet::GetVisibleRows( ); <b>void</b> CSpreadSheet::SetVisibleRows( <b>long</b> value);
Visual Basic	[form.]Spread.VisibleRows[ = value&]

### OCX Use

Number of Visible Rows box under Row Headers on the [Headers](#) property page

### Remarks

The default value for the VisibleRows property is 0.

Use the VisibleRows property when the [AutoSize](#) property is set to True. If the VisibleRows property is set to 0, the application uses the [MaxRows](#) property value.

### Data Type

Long Integer

[Print](#)

[Copy](#)

[Close](#)

The following example sets the number of visible rows to 4.

C++

```
// Set the number of visible rows  
vaSpread1->SetVisibleRows(4);
```

Visual Basic

```
' Set the number of visible rows  
vaSpread1.VisibleRows = 4
```

**See Also**

[AutoSize](#), [MaxRows](#), [VisibleCols](#) properties

## VScrollSpecial Property

[See Also](#)

[Example](#)

### Description

Sets or returns whether the spreadsheet uses a special vertical scroll bar.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetVScrollSpecial( ); <b>void</b> CSpreadSheet::SetVScrollSpecial( <b>BOOL</b> value);
Visual Basic	[form.]Spread.VScrollSpecial[ = boolean%]

### OCX Use

Use Special Scroll Bar check box under Vertical Scroll Bar on the [Virtual Mode](#) property page

### Remarks

Although virtual mode improves performance, it prevents the default scroll bar on the Spread control from accurately representing the position of the currently displayed record in the database. You can display a customized scroll bar by setting the VScrollSpecial property.

The VScrollSpecial property displays a special, customized vertical scroll bar instead of the default Windows scroll bar. Instead of a scroll box, this scroll bar has arrows the user clicks to go to the first page or last page, the previous or next page, or the previous or next line. You can remove some of these arrows by setting the [VScrollSpecialType](#) property.

This customized scroll bar works well with a Spread control in virtual mode (when the [VirtualMode](#) property is set to True). In virtual mode, the Spread control reads only as many database records as necessary to fill the list.

The default value for the VScrollSpecial property is False, which means the customized vertical scroll bar is not displayed.

### Data Type

Integer (Boolean)

**See Also**

[VirtualMode](#), [VScrollSpecialType](#) properties

[QueryData](#) event

## VScrollSpecialType Property

[See Also](#)

### Description

Sets or returns the scroll arrows to display on the special vertical scroll bar. This property is available at run time only.

### Syntax

```
C++          short CSpreadSheet::GetVScrollSpecialType( );  
            void CSpreadSheet::SetVScrollSpecialType(short value);  
  
Visual Basic  [form.]Spread.VScrollSpecialType[ = value%]
```

### OCX Use

No Home/End, No Page Up/Page Down, and No Line Up/Line Down check boxes under Vertical Scroll Bar on the [Virtual Mode](#) property page

### Remarks

Use the VScrollSpecialType property in conjunction with the VScrollSpecial property: the [VScrollSpecial](#) property turns on the scroll bar and the VScrollSpecialType property sets the style of scroll bar to display.

The following values are available. Note that you can combine values 1, 2, and 4 using the Or operator to customize display of the special scroll arrows. The default value is 0, which displays the home/end, page up/page down, and line up/line down arrows.

Value	Description	Constant
1	Does not display the home and end arrows	SS_VSCROLLSPECIAL_NO_HOME_END
2	Does not display the page up and page down arrows	SS_VSCROLLSPECIAL_NO_PAGE_UP_DOWN
4	Does not display the line up and line down arrows	SS_VSCROLLSPECIAL_NO_LINE_UP_DOWN

### Data Type

Integer

**See Also**

[VirtualMode](#), [VScrollSpecial](#) properties

[QueryData](#) event

---

## Width Property

[See Also](#)

### Description

Sets or returns the width of the Spread control.

### Syntax

C++	<b>long</b> CSpreadSheet::GetWidth( ); <b>void</b> CSpreadSheet::SetWidth( <b>long</b> value);
Visual Basic	[form.]Spread.Width[ = value!]

### OCX Use

The Width property is a standard extender property that can be provided by your container.

### Remarks

The value of this property changes as the user or the application changes the size of the control. The maximum value for this property is determined by the system.

The measurement for the width of the control is calculated from the center of the control's border so that controls with different border widths align correctly.

In Visual Basic, the measurement unit used by the Width property depends on the setting of the form's ScaleMode property. The default ScaleMode setting is twips (1/1440 of an inch). C++ uses pixels as the measurement unit.

Visual Basic users can refer to the Visual Basic documentation for additional information about this property.

### Data Type

Single



**See Also**

[Height](#) property

## Events

[Advance](#)  
[BlockSelected](#)  
[ButtonClicked](#)  
[Change](#)  
[Click](#)  
[ColWidthChange](#)  
[CustomFunction](#)  
[DataAddNew](#)  
[DataColConfig](#)  
[DataFill](#)  
[DbClick](#)  
[DragDrop](#)  
[DragDropBlock](#)  
[DragOver](#)  
[DrawItem](#)  
[EditError](#)  
[EditMode](#)  
[EnterRow](#)  
[GotFocus](#)  
[KeyDown](#)  
[KeyPress](#)  
[KeyUp](#)  
[LeaveCell](#)  
[LeaveRow](#)  
[LostFocus](#)  
[MouseDown](#)  
[MouseMove](#)  
[MouseUp](#)  
[PrintAbort](#)  
[QueryAdvance](#)  
[QueryData](#)  
[RightClick](#)  
[RowHeightChange](#)  
[SelChange](#)  
[TopLeftChange](#)  
[UserFormulaEntered](#)  
[VirtualClearData](#)

## Advance Event

[See Also](#)

### Description

Occurs when the user performs one of the following actions:

- Presses the Tab key when the focus is in the last cell
- Presses Shift+Tab when the focus is in the first cell
- Presses the Down Arrow key when the focus is in the last row
- Presses the Up Arrow key when the focus is in the first row

### Syntax

C++ (MFC)	<code>afx_msg void OnAdvanceSpread(<b>UINT</b>, <b>int</b>, <b>CWnd*</b>, ↳<b>LPVOID</b>);</code>
C++ (OWL)	<code>void EvAdvance(<b>VBXEVENT far*</b> event);</code>
Visual Basic (VBX)	<code>Sub Spread_Advance(AdvanceNext As <b>Integer</b>)</code>
Visual Basic (OCX)	<code>Sub Spread_Advance(ByVal AdvanceNext As <b>Boolean</b>)</code>

### Parameter

The following parameter is available:

Parameter	Description
<i>AdvanceNext</i>	Specifies whether the user is moving to the next control or the previous control

### Remarks

You must set the [ProcessTab](#) property to True for the Advance event to occur.

If the *AdvanceNext* parameter is True, the user pressed the Tab or Down Arrow key. If it is False, the user pressed Shift+Tab or the Up Arrow key.

**See Also**

[ProcessTab](#) property

## BlockSelected Event

[See Also](#)

### Description

Occurs when the user completes the selection of a block of cells.

### Syntax

C++ (MFC)	<code>afx_msg void OnBlockSelectedSpread(UINT, int, CWnd*, ↳LPVOID);</code>
C++ (OWL)	<code>void EvBlockSelected(VBXEVENT far* event);</code>
Visual Basic (VBX)	<code>Sub Spread_BlockSelected(BlockCol As Long, BlockRow As Long, ↳BlockCol2 As Long, BlockRow2 As Long)</code>
Visual Basic (OCX)	<code>Sub Spread_BlockSelected(ByVal BlockCol As Long, ↳ByVal BlockRow As Long,</code>

`↳ByVal BlockCol2 As Long,`

`↳ByVal BlockRow2 As Long)`

### Parameters

The following parameters are available:

Parameter	Description
<i>BlockCol</i>	Column number of cell in upper-left corner of selected block
<i>BlockRow</i>	Row number of cell in upper-left corner of selected block
<i>BlockCol2</i>	Column number of cell in lower-right corner of selected block
<i>BlockRow2</i>	Row number of cell in lower-right corner of selected block

### See Also

[IsBlockSelected](#), [SelBlockCol](#), [SelBlockCol2](#), [SelBlockRow](#), [SelBlockRow2](#), [SelectBlockOptions](#) properties  
[SelChange](#) event

## ButtonClicked Event

[See Also](#)

### Description

Occurs when the user clicks a button cell or check box cell.

### Syntax

C++ (MFC)	<code>afx_msg void OnButtonClickedSpread(<b>UINT</b>, <b>int</b>, <b>CWnd*</b>, ↳<b>LPVOID</b>);</code>
C++ (OWL)	<code>void EvButtonClicked(<b>VBXEVENT far*</b> event);</code>
Visual Basic (VBX)	<code>Sub Spread_ButtonClicked(<b>Col</b> As <b>Long</b>, <b>Row</b> As <b>Long</b>, ↳<b>ButtonDown</b> As <b>Integer</b>)</code>
Visual Basic (OCX)	<code>Sub Spread_ButtonClicked(ByVal <b>Col</b> As <b>Long</b>, ↳ByVal <b>Row</b> As <b>Long</b>,</code>

—ByVal *ButtonDown* As **Integer**)

### Parameters

The following parameters are available:

Parameter	Description
<i>Col</i>	Column number of cell the user clicked
<i>Row</i>	Row number of cell the user clicked
<i>ButtonDown</i>	Value of a two-state button (True or False) (Parameter is always True if button is not a two-state button.)

### Remarks

When the [TypeButtonType](#) property is set to 0 (Normal), this event occurs when the user releases the mouse button while the mouse pointer is over the button cell.

When the [TypeButtonType](#) property is set to 1 (Two-State), this event occurs when the user presses a mouse button while the mouse pointer is over the button cell. The event also occurs when the user releases the mouse button.

The *ButtonDown* parameter contains the state of a two-state button. The value is True if the button is down and False if the button is up. The *ButtonDown* parameter is useful only when the [TypeButtonType](#) property is set to 1 (Two-State).

**See Also**

[TypeButtonType](#) property

## Change Event

[See Also](#)

### Description

Occurs when the user modifies the contents of a cell.

### Syntax

C++ (MFC)	<code>afx_msg void OnChangeSpread(UINT, int, CWnd*, LPVOID);</code>
C++ (OWL)	<code>void EvChange(VBXEVENT far* event);</code>
Visual Basic (VBX)	<code>Sub Spread_Change(Col As Long, Row As Long)</code>
Visual Basic (OCX)	<code>Sub Spread_Change(ByVal Col As Long, ByVal Row As Long)</code>

### Parameters

The following parameters are available:

Parameter	Description
<i>Col</i>	Column number of cell with change
<i>Row</i>	Row number of cell with change

### Remarks

This event also occurs if any dependent cell in a formula changes. For example, cells A1, B1, and C1 are integer cells. Cell C1 contains the formula A1 + B1. If the user changes cell A1, two Change events will occur, one for cell A1 and the other for cell C1.

Use the [Text](#) or [Value](#) property to retrieve the text in the changed cell. If the user changes a cell, the Change event is sent when the user leaves the cell.

Note that the Change event does not occur if the application changes the contents of a cell.



**See Also**

[Text](#), [Value](#) properties

## Click Event

[See Also](#)

### Description

Occurs when the user clicks a cell using the left mouse button.

### Syntax

C++ (MFC)	<code>afx_msg void OnClickSpread(<b>UINT</b>, <b>int</b>, <b>CWnd*</b>, <b>LPVOID</b>);</code>
C++ (OWL)	<code>void EvClick(<b>VBXEVENT far*</b> event);</code>
Visual Basic (VBX)	<code>Sub Spread_Click(<b>Col</b> As <b>Long</b>, <b>Row</b> As <b>Long</b>)</code>
Visual Basic (OCX)	<code>Sub Spread_Click(ByVal <b>Col</b> As <b>Long</b>, ByVal <b>Row</b> As <b>Long</b>)</code>

### Parameters

The following parameters are available:

Parameter	Description
<i>Col</i>	Column number of cell the user clicked
<i>Row</i>	Row number of cell the user clicked

### See Also

[DbClick](#), [RightClick](#) events

## ColWidthChange Event

[See Also](#)

### Description

Occurs when the user changes the column width using the mouse.

### Syntax

C++ (MFC)	<code>afx_msg void OnColWidthChangeSpread(<b>UINT</b>, <b>int</b>, <b>Cwnd*</b>, —<b>LPVOID</b>);</code>
C++ (OWL)	<code>void EvColWidthChange(<b>VBXEVENT far*</b> event);</code>
Visual Basic (VBX)	<code>Sub <i>Spread_ColWidthChange</i>(<b>Col1</b> As <b>Long</b>, <b>Col2</b> As <b>Long</b>)</code>
Visual Basic (OCX)	<code>Sub <i>Spread_ColWidthChange</i>(ByVal <b>Col1</b> As <b>Long</b>, —ByVal <b>Col2</b> As <b>Long</b>)</code>

### Parameters

The following parameters are available:

Parameter	Description
<i>Col1</i>	Column number of first column whose width changed
<i>Col2</i>	Column number of last column whose width changed

### Remarks

In most cases, the column value for the *Col2* parameter is the same as the value of the *Col1* parameter. If the value is larger, the widths of the range of columns specified by *Col1* and *Col2* have all changed.

**See Also**

[ColWidth](#), [UserResize](#), [UserResizeCol](#) properties

[RowHeightChange](#) event

## CustomFunction Event

[See Also](#)

[C++ \(MFC\) Example](#)

[C++ \(OWL\) Example](#)

[Visual Basic Example](#)

### Description

Occurs when a user-defined function is evaluated in a cell's formula.

### Syntax

C++ (MFC)            `afx_msg void OnCustomFunctionSpread(UINT, int, CWnd*,  
—LPVOID);`

C++ (OWL)            `void EvCustomFunction(VBXEVENT far* event);`

Visual Basic (VBX)   `Sub Spread_CustomFunction(FunctionName As String,  
—ParameterCnt As Integer, Col As Long,`

`—Row As Long, Status As Integer)`

Visual Basic (OCX)   `Sub Spread_CustomFunction(ByVal FunctionName As String,  
—ByVal ParameterCnt As Integer,`

`—ByVal Col As Long, ByVal Row As Long,`

`—Status As Integer)`

### Parameters

The following parameters are available:

Parameter	Description
<i>FunctionName</i>	Name of function being evaluated
<i>ParameterCnt</i>	Number of parameters passed to function
<i>Col</i>	Column number of cell that contains function
<i>Row</i>	Row number of cell that contains function
<i>Status</i>	Return value from function evaluation Can be one of the following: SS_VALUE_STATUS_OK SS_VALUE_STATUS_ERROR SS_VALUE_STATUS_EMPTY

[Print](#)

[Copy](#)

[Close](#)

C++ (OWL)

The following example implements a "cube(x)" function.

```
void TDemoWnd::EvCustomFunction(VBXEVENT far* event)
{
    HLSTR hlstrFuncName = SS_CUSTOMFUNCTION_FUNCTIONNAME
        (event);
    short nParamCnt = SS_CUSTOMFUNCTION_PARAMETERCNT(event);
    long lCol = SS_CUSTOMFUNCTION_COL(event);
    long lRow = SS_CUSTOMFUNCTION_ROW(event);
    short nStatus = SS_VALUE_STATUS_EMPTY;
    LPSTR lpszFuncName = VBXGetBasicStringPtr(hlstrFuncName);
    short nFuncNameLen = VBXGetBasicStringLength
        (hlstrFuncName);

    if( _fstrnicmp("cube", lpszFuncName, nFuncNameLen) == 0 )
    {
        short nArgType;
        short nArgStatus;
        pSpread->CFGetParamInfo(1, &nArgType, &nArgStatus);
        if( SS_VALUE_STATUS_OK == nArgStatus )
        {
            if( SS_VALUE_TYPE_LONG == nArgType )
```

```
{
    long lArg, lResult;
    lArg = pSpread->CFGetLongParam(1);
    lResult = lArg * lArg * lArg;
    pSpread->CFSetResult(lResult);
    nStatus = SS_VALUE_STATUS_OK;
}
else if( SS_VALUE_TYPE_DOUBLE == nArgType )
{
    double dfArg, dfResult;
    pSpread->CFGetDoubleParamExt(1, &dfArg);
    dfResult = dfArg * dfArg * dfArg;
    pSpread->CFSetResult(dfResult);
    nStatus = SS_VALUE_STATUS_OK;
}
}
}
SS_CUSTOMFUNCTION_STATUS(event) = nStatus;
}
```

[Print](#)

[Copy](#)

[Close](#)

#### Visual Basic

```
Private Sub vaSpread1_CustomFunction(ByVal FunctionName
    As String, ByVal ParameterCnt As Integer, ByVal Col
    As Long, ByVal Row As Long, Status As Integer)
    Dim vResult As Variant
    Dim b As Boolean
    Dim lArg1 As Long
    Dim dfArg1 As Double
    Dim nTypeArg1 As Integer
    Dim nStatusArg1 As Integer
    ' Return a status of empty when a result cannot be
    ' computed
    Status = SS_VALUE_STATUS_EMPTY
    ' Implement the "cube()" custom function
    If FunctionName = "cube" Then
        b = vaSpread1.CFGetParamInfo(1, nTypeArg1, nStatusArg1)
        If nStatusArg1 = SS_VALUE_STATUS_OK Then
            If nTypeArg1 = SS_VALUE_TYPE_LONG Then
                lArg1 = vaSpread1.CFGetLongParam(1)
                vResult = lArg1 * lArg1 * lArg1
                Status = SS_VALUE_STATUS_OK
            ElseIf nTypeArg1 = SS_VALUE_TYPE_DOUBLE Then
                dfArg1 = vaSpread1.CFGetDoubleParam(1)
                vResult = dfArg1 * dfArg1 * dfArg1
                Status = SS_VALUE_STATUS_OK
            End If
        End If
    End If
    If Status = SS_VALUE_STATUS_OK Then
        b = vaSpread1.CFSetResult(vResult)
    End If
End Sub
```

**See Also**

[Formula](#) property

[AddCustomFunction](#), [CFGetDoubleParam](#), [CFGetLongParam](#), [CFGetParamInfo](#), [CFGetStringParam](#), [CFSetResult](#) functions



## DataAddNew Event

### Description

Occurs when the Spread control is bound to an empty database table and a record is inserted. This event occurs in Visual Basic only.

**Note** Call the Visual Basic database AddNew method to insert the record.

### Syntax

Visual Basic            Sub *Spread\_DataAddNew*( *void* )

## DataColConfig Event

[See Also](#)

### Description

Occurs for each column in the spreadsheet as the spreadsheet is bound to a database.

### Syntax

```
C++ (MFC)      afx_msg void OnDataColConfigSpread(UINT, int, CWnd*,
               —LPVOID);
C++ (OWL)      void EvDataColConfig(VBXEVENT far* event);
Visual Basic (VBX)  Sub Spread_DataColConfig(Col As Long, DataField As String,
               —DataType As Integer)
Visual Basic (OCX)  Sub Spread_DataColConfig(ByVal Col As Long,
               —ByVal DataField As String,
```

—ByVal DataType As Integer)

### Parameters

The following parameters are available:

Parameter	Description
<i>Col</i>	Column number of column being configured
<i>DataField</i>	Database field name
<i>DataType</i>	Data type as defined by the database (See the table of data types in the following Remarks section.)

### Remarks

This event occurs for bound Spread controls. After the control is bound to a database, if the [DAutoCellTypes](#) property is set to True, the Spread control sets the cell type for each cell based on the type of data in the database field bound to the cell. If the [DAutoSizeCols](#) property is set to True, this event occurs before the columns automatically size for the data.

You can evaluate the cell types set by the Spread control for the bound data and change them as needed within the event procedure. Use the [CellType](#) property to determine the assigned cell type; then change the cell types as appropriate.

The following data types are assigned when the DAutoCellTypes property is set to True:

Data Type	Spread Cell Type	Notes
1 - Boolean	Check Box	Displays selected check box when value is 0; displays cleared check box when value is 1
2 - Byte	Integer	Minimum value: 0 Maximum value: 255
3 - Integer	Integer	Minimum value: —32,768 Maximum value: 32,767
4 - Long	Integer	Minimum value: —2,147,483,647 Maximum value: 2,147,483,647
5 - Currency	Float	Minimum value: —922,337,203,685,477.5808 Maximum value: 922,337,203,685,477.5807
6 - Single	Float	Minimum and maximum values are determined by your development environment
7 - Double	Float	Minimum and maximum values are determined by your development environment
8 - Date/Time	Date	Displays only the date (You can change the cell type to Time, if you prefer.)
10 - Text	Edit	ASCII text with a maximum length of 64K (The string should not include embedded tabs.)
11 - Binary	Picture	Displays only bitmaps
12 - Memo	Edit	Displays multiple-line edit cell

### See Also

[CellType](#), [DAutoCellTypes](#), [DAutoSizeCols](#) properties

## DataFill Event

[See Also](#)

### Description

Occurs for each cell in the Spread control before it receives data from the database and before it writes data to the database, if the [DataFillEvent](#) property is set to True for the cell.

### Syntax

C++ (MFC)                    `afx_msg void OnDataFillSpread(UINT, int, CWnd*, LPVOID);`

C++ (OWL)                    `void EvDataFill(VBxEVENT far* event);`

Visual Basic (VBX)        `Sub Spread_DataFill(Col As Long, Row As Long,  
                          —DataType As Integer, fGetData As Integer,`

—Cancel As Integer)

Visual Basic (OCX)        `Sub Spread_DataFill(ByVal Col As Long, ByVal Row As Long,  
                          —ByVal DataType As Integer,`

—ByVal fGetData As Integer, Cancel As Integer)

### Parameters

The following parameters are available:

Parameter	Description
<i>Col</i>	Column number of cell assigned data
<i>Row</i>	Row number of cell assigned data
<i>DataType</i>	Data type as defined by the database (See the <a href="#">DataColConfig</a> event for a table of data types.)
<i>fGetData</i>	True if the Spread control is retrieving data; False if data is being written to the database
<i>Cancel</i>	Set to True to prevent default processing (The Spread control will not automatically read or write data.)

### Remarks

This event occurs for bound Spread controls. This event occurs only for cells whose column has the [DataFillEvent](#) property set to True.

For each cell, this event occurs before data is assigned to the specified cell in the Spread control and before the Spread control writes data from the cell to the database, so the application can provide special formatting.

To retrieve the value of the data being assigned to the cell, use the [GetDataFillData](#) function. To modify the data being written to the database, use the [SetDataFillData](#) function.

**Note** The *DataType* parameter represents the data access type. The *Vtype* (VBX) and *VarType* (OCX) parameters for the [GetDataFillData](#) function represent a variant data type. These parameter values are not equivalent. For example, the [DataFill](#) event would return a value of 4 in the *DataType* parameter for a long integer. For the [GetDataFillData](#) function, the variant data type for a long integer is represented by a value of 3.

**See Also**

[DataFillEvent](#) property

[GetDataFillData](#), [SetDataFillData](#) functions

## DblClick Event

[See Also](#)

### Description

Occurs when the user double-clicks the left mouse button while the mouse pointer is in a cell.

### Syntax

C++ (MFC)	<code>afx_msg void OnDblClickSpread(<b>UINT</b>, <b>int</b>, <b>CWnd*</b>, —<b>LPVOID</b>);</code>
C++ (OWL)	<code>void EvDblClick(<b>VBXEVENT far*</b> event);</code>
Visual Basic (VBX)	<code>Sub Spread_DblClick(<b>Col</b> As <b>Long</b>, <b>Row</b> As <b>Long</b>)</code>
Visual Basic (OCX)	<code>Sub Spread_DblClick(ByVal <b>Col</b> As <b>Long</b>, ByVal <b>Row</b> As <b>Long</b>)</code>

### Parameters

The following parameters are available:

Parameter	Description
<i>Col</i>	Column number of cell the user double-clicked
<i>Row</i>	Row number of cell the user double-clicked

### See Also

[Click](#), [RightClick](#) events

## DragDrop Event

[See Also](#)

### Description

Occurs when a drag-and-drop operation is completed as a result of dragging a control over a form or control and releasing the mouse button. An object can be a source (an item the user drags) or a target (an item on which the user drops a source).

### Syntax

C++ (MFC)	<code>afx_msg void OnDragDropSpread(<b>UINT</b>, <b>int</b>, <b>CWnd*</b>, <b>LPVOID</b>);</code>
C++ (OWL)	<code>void EvDragDrop(<b>VBXEVENT far*</b> event);</code>
Visual Basic	<code>Sub Spread_DragDrop([<b>Index As Integer</b>,] <b>Source As Control</b>, —<b>X As Single</b>, <b>Y As Single</b>)</code>

### OCX Use

The DragDrop event is a standard extender event that can be provided by your container.

### Parameters

The following parameters are available:

Parameter	Description
<i>Index</i>	Identifies a control if it is in an array
<i>Source</i>	The control being dragged (You can refer to properties with this argument, for example, <code>vaSpread1.Visible = 0.</code> )
<i>X, Y</i>	The current horizontal ( <i>X</i> ) and vertical ( <i>Y</i> ) position of the mouse pointer within the target form or control (These coordinates are always expressed in terms of the target's coordinate system.)

### Remarks

Use a DragDrop event to control what happens after a drag operation has been completed. For example, you can move the source control to a new location or copy a button from one location to another.

**Note** Use the [DragMode](#) property and Drag method to specify the way dragging is initiated. Once dragging has been initiated, you can handle events that precede a DragDrop event with a [DragOver](#) event.

Visual Basic users can refer to the Visual Basic documentation for additional information about this event.

**See Also**

[DragIcon](#), [DragMode](#) properties

[DragOver](#), [MouseDown](#), [MouseMove](#), [MouseUp](#) events



## DragDropBlock Event

[See Also](#)

### Description

Occurs when the user drags and drops a block of cells to a new location.

### Syntax

C++ (MFC)                   afx\_msg **void** OnDragDropBlockSpread(**UINT**, **int**, **CWnd\***, **LPVOID**);

C++ (OWL)                   **void** EvDragDropBlock(**VBXEVENT far\*** event);

Visual Basic (VBX)       Sub *Spread\_DragDropBlock*(*Col* As **Long**, *Row* As **Long**,  
                                  — *Col2* As **Long**, *Row2* As **Long**, *NewCol* As **Long**,

— *NewRow* As **Long**, *NewCol2* As **Long**,

— *NewRow2* As **Long**, *Overwrite* As **Integer**,

— *Action* As **Integer**, *DataOnly* As **Integer**,

— *Cancel* As **Integer**)

Visual Basic (OCX)       Sub *Spread\_DragDropBlock*(ByVal *Col* As **Long**, ByVal *Row* As **Long**,  
                                  — ByVal *Col2* As **Long**, ByVal *Row2* As **Long**, ByVal *NewCol* As **Long**,

— ByVal *NewRow* As **Long**, ByVal *NewCol2* As **Long**, ByVal *NewRow2*

— As **Long**, ByVal *Overwrite* As **Boolean**, *Action* As **Integer**,

— *DataOnly* As **Boolean**, *Cancel* As **Boolean**)

### Parameters

The following parameters are available:

Parameter	Description
<i>Col</i>	Column number of leftmost column in block
<i>Row</i>	Row number of top row in block
<i>Col2</i>	Column number of rightmost column in block
<i>Row2</i>	Row number of bottom row in block
<i>NewCol</i>	Column number of leftmost column in destination block
<i>NewRow</i>	Row number of top row in destination block
<i>NewCol2</i>	Column number of rightmost column in destination block
<i>NewRow2</i>	Row number of bottom row in destination block
<i>Overwrite</i>	Value is nonzero if destination block contains data
<i>Action</i>	Value is 0 if the block is being moved or 1 if the block is being copied
<i>DataOnly</i>	Set this value to True to copy or move only data before exiting the event
<i>Cancel</i>	Set this value to True to cancel the drag-and-drop operation before exiting the event

### Remarks

The user can only perform drag-and-drop operations when the [AllowDragDrop](#) property is set to True.

As indicated in the parameter table, the event procedure can set the *Action*, *DataOnly*, and *Cancel* parameters. By default, the Spread control moves data and cell attributes in a drag-and-drop operation. You can change the value of the *Action* parameter to force a copy operation. Change the value of the *DataOnly* parameter to copy or move only data. Use the *Cancel* parameter to cancel the operation.

**See Also**

[AllowDragDrop](#) property

## DragOver Event

[See Also](#)

### Description

Occurs when a drag-and-drop operation is in process.

### Syntax

C++ (MFC)	<code>afx_msg void OnDragOverSpread(UINT, int, CWnd*, LPVOID);</code>
C++ (OWL)	<code>void EvDragOver(VBXEVENT far* event);</code>
Visual Basic	<code>Sub Spread_DragOver([Index As Integer,] Source As Control, —X As Single, Y As Single, State As Integer)</code>

### OCX Use

The DragOver event is a standard extender event that can be provided by your container.

### Parameters

The following parameters are available:

Parameter	Description
<i>Index</i>	Identifies a control if it is in an array
<i>Source</i>	Identifies the current target control
<i>X, Y</i>	The current horizontal ( <i>X</i> ) and vertical ( <i>Y</i> ) position of the mouse pointer within the target form or control (These coordinates are always expressed in terms of the target's coordinate system.)
<i>State</i>	The transition state of the control being dragged in relation to a target form or control: 0 - Enter      Source control is being dragged within the range of a target 1 - Leave      Source control is being dragged out of the range of a target 2 - Over      Source control has moved from one position in the target to another

### Remarks

Use a DragOver event to monitor when the mouse pointer enters, leaves, or is directly over a valid target. The mouse pointer position determines which target object receives this event.

**Note** Use the [DragMode](#) property and Drag method to specify the way dragging is initiated. Once dragging has been initiated, you can handle events that precede a [DragDrop](#) event with a DragOver event.

Visual Basic users can refer to the Visual Basic documentation for additional information about this event.

**See Also**

[DragIcon](#), [DragMode](#) properties

[DragDrop](#), [MouseDown](#), [MouseMove](#), [MouseUp](#) events

## DrawItem Event

[See Also](#)

### Description

Occurs when the Spread control displays an owner-drawn cell.

### Syntax

C++ (MFC)                `afx_msg void OnDrawItemSpread(UINT, int, CWnd*, LPVOID);`  
C++ (OWL)                `void EvDrawItem(VBXEVENT far* event);`  
Visual Basic (VBX)        `Sub Spread_DrawItem(Col As Long, Row As Long, hDC As Integer,`  
                              `—Left As Long, Top As Long, Right As Long,`

`—Bottom As Long, Style As Long)`

Visual Basic (OCX)        `Sub Spread_DrawItem(ByVal Col As Long, ByVal Row As Long,`  
                              `—ByVal hDC As Integer, ByVal Left As Long,`

`—ByVal Top As Long, ByVal Right As Long,`

`—ByVal Bottom As Long, ByVal Style As Long)`

### Parameters

The following parameters are available:

Parameter	Description
<i>Col</i>	Column number of cell being drawn
<i>Row</i>	Row number of cell being drawn
<i>hDC</i>	Device context handle of the Spread control (This must be used for all painting to the cell.)
<i>Left</i>	Left screen coordinate of the cell being drawn relative to the upper-left corner of the Spread control (This value is in twips.)
<i>Top</i>	Top screen coordinate of the cell being drawn relative to the upper-left corner of the Spread control (This value is in twips.)
<i>Right</i>	Right screen coordinate of the cell being drawn relative to the upper-left corner of the Spread control (This value is in twips.)
<i>Bottom</i>	Bottom screen coordinate of the cell being drawn relative to the upper-left corner of the Spread control (This value is in twips.)
<i>Style</i>	Style of the cell being drawn (This value is assigned by the application using the <a href="#">TypeOwnerDrawStyle</a> property.)

### See Also

[TypeOwnerDrawStyle](#) property

## EditError Event

[See Also](#)

### Description

Occurs when the user performs an invalid input operation.

### Syntax

C++ (MFC)            `afx_msg void OnEditErrorSpread(UINT, int, CWnd*, LPVOID);`  
C++ (OWL)            `void EvEditError(VBXEVENT far* event);`  
Visual Basic (VBX)    `Sub Spread_EditError(Col As Long, Row As Long,  
                          —EditError As Integer)`  
Visual Basic (OCX)    `Sub Spread_EditError(ByVal Col As Long, ByVal Row As Long,  
                          —ByVal EditError As Integer)`

### Parameters

The following parameters are available:

Parameter	Description																																								
<i>Col</i>	Column number of cell with error																																								
<i>Row</i>	Row number of cell with error																																								
<i>EditError</i>	Error that occurred One of the following codes: <table><thead><tr><th>Error Code</th><th>Description</th></tr></thead><tbody><tr><td>1</td><td>User attempted to interact with a locked cell</td></tr><tr><td>2</td><td>User attempted to move the focus to a restricted column</td></tr><tr><td>3</td><td>User attempted to move the focus to a restricted row</td></tr><tr><td>4</td><td>User attempted to move the focus to a cell that is located in both a restricted row and column</td></tr><tr><td>5</td><td>User typed invalid data into a cell (The event is sent when the user leaves the cell.)</td></tr><tr><td>9</td><td>User attempted to type in a check box cell</td></tr><tr><td>100</td><td>An error occurred during a BeginUpdate-Add function call to the Data Access Object</td></tr><tr><td>101</td><td>An error occurred during a BeginUpdate-Edit function call to the Data Access Object</td></tr><tr><td>102</td><td>An error occurred during a Delete function call to the Data Access Object</td></tr><tr><td>103</td><td>An error occurred during a SetColumn function call to the Data Access Object</td></tr><tr><td>104</td><td>An error occurred during an Update function call to the Data Access Object</td></tr><tr><td>105</td><td>An invalid value was found in the data before it was written to the database</td></tr><tr><td>106</td><td>User deleted a row; therefore, the current row is no longer valid</td></tr><tr><td>107</td><td>A block of memory could not be allocated</td></tr><tr><td>108</td><td>Current operation is not supported in Visual Basic 4.0</td></tr><tr><td>109</td><td>Current row could not be updated due to insufficient access privileges</td></tr><tr><td>110</td><td>User attempted to update a row that has not been updated in the database (To resolve this error, refresh the database.)</td></tr><tr><td>111</td><td>User attempted to move to a row that did not exist</td></tr><tr><td>112</td><td>Database operation failed</td></tr></tbody></table>	Error Code	Description	1	User attempted to interact with a locked cell	2	User attempted to move the focus to a restricted column	3	User attempted to move the focus to a restricted row	4	User attempted to move the focus to a cell that is located in both a restricted row and column	5	User typed invalid data into a cell (The event is sent when the user leaves the cell.)	9	User attempted to type in a check box cell	100	An error occurred during a BeginUpdate-Add function call to the Data Access Object	101	An error occurred during a BeginUpdate-Edit function call to the Data Access Object	102	An error occurred during a Delete function call to the Data Access Object	103	An error occurred during a SetColumn function call to the Data Access Object	104	An error occurred during an Update function call to the Data Access Object	105	An invalid value was found in the data before it was written to the database	106	User deleted a row; therefore, the current row is no longer valid	107	A block of memory could not be allocated	108	Current operation is not supported in Visual Basic 4.0	109	Current row could not be updated due to insufficient access privileges	110	User attempted to update a row that has not been updated in the database (To resolve this error, refresh the database.)	111	User attempted to move to a row that did not exist	112	Database operation failed
Error Code	Description																																								
1	User attempted to interact with a locked cell																																								
2	User attempted to move the focus to a restricted column																																								
3	User attempted to move the focus to a restricted row																																								
4	User attempted to move the focus to a cell that is located in both a restricted row and column																																								
5	User typed invalid data into a cell (The event is sent when the user leaves the cell.)																																								
9	User attempted to type in a check box cell																																								
100	An error occurred during a BeginUpdate-Add function call to the Data Access Object																																								
101	An error occurred during a BeginUpdate-Edit function call to the Data Access Object																																								
102	An error occurred during a Delete function call to the Data Access Object																																								
103	An error occurred during a SetColumn function call to the Data Access Object																																								
104	An error occurred during an Update function call to the Data Access Object																																								
105	An invalid value was found in the data before it was written to the database																																								
106	User deleted a row; therefore, the current row is no longer valid																																								
107	A block of memory could not be allocated																																								
108	Current operation is not supported in Visual Basic 4.0																																								
109	Current row could not be updated due to insufficient access privileges																																								
110	User attempted to update a row that has not been updated in the database (To resolve this error, refresh the database.)																																								
111	User attempted to move to a row that did not exist																																								
112	Database operation failed																																								

**Note** Error codes 100 through 112 apply to OCX controls only.

**See Also**

[DataColCnt](#), [DataRowCnt](#), [Lock](#), [Protect](#), [RestrictCols](#), [RestrictRows](#) properties

## EditMode Event

[See Also](#)

### Description

Occurs when the user enters or leaves edit mode in a cell.

### Syntax

C++ (MFC)            `afx_msg void OnEditModeSpread(UINT, int, CWnd*, LPVOID);`  
C++ (OWL)            `void EvEditMode(VBXEVENT far* event);`  
Visual Basic (VBX)   `Sub Spread_EditMode(Col As Long, Row As Long, Mode As Integer,`  
                         `—ChangeMade As Integer)`  
Visual Basic (OCX)   `Sub Spread_EditMode(ByVal Col As Long, ByVal Row As Long,`  
                         `—ByVal Mode As Integer,`

`—ByVal ChangeMade As Boolean)`

### Parameters

The following parameters are available:

Parameter	Description						
<i>Col</i>	Column number of active cell						
<i>Row</i>	Row number of active cell						
<i>Mode</i>	Either of the following: <table><thead><tr><th>Mode Value</th><th>Description</th></tr></thead><tbody><tr><td>0</td><td>Edit mode is off (User pressed Enter key, another cell was activated, or the application lost the focus.)</td></tr><tr><td>1</td><td>Edit mode is on (User started typing in the cell or double-clicked the cell, or the <a href="#">EditMode</a> property was set to True.)</td></tr></tbody></table>	Mode Value	Description	0	Edit mode is off (User pressed Enter key, another cell was activated, or the application lost the focus.)	1	Edit mode is on (User started typing in the cell or double-clicked the cell, or the <a href="#">EditMode</a> property was set to True.)
Mode Value	Description						
0	Edit mode is off (User pressed Enter key, another cell was activated, or the application lost the focus.)						
1	Edit mode is on (User started typing in the cell or double-clicked the cell, or the <a href="#">EditMode</a> property was set to True.)						
<i>ChangeMade</i>	If edit mode is off and this value is True, the user changed the text in the active cell						

### See Also

[EditMode](#), [EditModePermanent](#), [EditModeReplace](#) properties



## EnterRow Event

### Description

Occurs when the operation mode is set to row mode and the focus moves to a new row.

### Syntax

```
C++ (MFC)          afx_msg void OnEnterRowSpread(UINT, int, CWnd*, LPVOID);
C++ (OWL)          void EvEnterRow(VBXEVENT far* event);
Visual Basic (VBX) Sub Spread_EnterRow(Row As Long, RowsLast As Integer)
Visual Basic (OCX) Sub Spread_EnterRow(ByVal Row As Long,
—ByVal RowsLast As Long)
```

### Parameters

The following parameters are available:

Parameter	Description
<i>Row</i>	Row number of row receiving the focus
<i>RowsLast</i>	Value is nonzero if row receiving the focus is beyond last row containing data

## GotFocus Event

[See Also](#)

### Description

Occurs when an object receives the focus, either by a user action such as tabbing to or clicking the object, or by changing the focus in code using the SetFocus method.

### Syntax

```
C++ (MFC)          afx_msg void OnGotFocusSpread(UINT, int, CWnd*, LPVOID);
C++ (OWL)          void EvGotFocus(VBXEVENT far* event);
Visual Basic       Sub Spread_GotFocus([Index As Integer])
```

### OCX Use

The GotFocus event is a standard extender event that can be provided by your container.

### Parameter

The following parameter is available:

Parameter	Description
<i>Index</i>	Identifies a control if it is in an array

### Remarks

Use a GotFocus event to specify the actions that occur when a control or form first receives the focus. For example, by attaching a GotFocus event to each control on a form, you can guide the user by displaying brief instructions or status bar messages. You can also provide visual cues by enabling, disabling, or showing other controls that depend on the control that has the focus.

**Note** An object can receive the focus only if its [Enabled](#) or [Visible](#) properties are set to True.

Visual Basic users can refer to the Visual Basic documentation for additional information about this event.

**See Also**

[TabIndex](#) property

## KeyDown, KeyUp Events

### Description

Occur when the user presses (KeyDown) or releases (KeyUp) a key while an object has the focus.

### Syntax

C++ (MFC)	<code>afx_msg void OnKeyDownSpread(UINT, int, CWnd*, LPVOID);</code> <code>afx_msg void OnKeyUpSpread(UINT, int, CWnd*, LPVOID);</code>
C++ (OWL)	<code>void EvKeyDown(VBXEVENT far* event);</code> <code>void EvKeyUp(VBXEVENT far* event);</code>
Visual Basic	<code>Sub Spread_KeyDown([Index As Integer,] KeyCode As Integer,</code> <code>— Shift As Integer)</code>  <code>Sub Spread_KeyUp([Index As Integer,] KeyCode As Integer,</code> <code>— Shift As Integer)</code>

### OCX Use

The KeyDown and KeyUp events are stock events.

### Parameters

The following parameters are available:

Parameter	Description
<i>Index</i>	Identifies a control if it is in an array
<i>KeyCode</i>	A key code
<i>Shift</i>	The state of the Shift, Ctrl, and Alt keys at the time of the event (The <i>Shift</i> parameter is a bit field, with the least-significant bits corresponding to the Shift key (bit 0), the Ctrl key (bit 1), and the Alt key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both Ctrl and Alt are pressed, the value of <i>Shift</i> is 6.)

### Remarks

Use the KeyDown and KeyUp events for keyboard handlers if you need to respond to both the pressing and releasing of a key. For both events, the object with the focus receives all keystrokes.

Although the KeyDown and KeyUp events can apply to most keys, they are most often used for

- Extended character keys such as function keys (F1
- F16)
- Navigation keys (Home, End, Page Up, Page Down, Up Arrow, Down Arrow, Right Arrow, Left Arrow, and Tab keys)
- Combinations of keys with the Alt, Ctrl, and Shift keys
- Distinguishing between the numeric keypad and regular number keys

The KeyDown and KeyUp events might not apply to keys in some situations. For example, when a default or Cancel command button is on a form, the events do not apply to the Esc, Enter, or Tab keys.

Visual Basic users can refer to the Visual Basic documentation for additional information about this event.

# KeyPress Event

## Description

Occurs when the user presses and releases an ANSI key.

## Syntax

C++ (MFC)            `afx_msg void OnKeyPressSpread(UINT, int, CWnd*, LPVOID);`  
C++ (OWL)            `void EvKeyPress(VBXEVENT far* event);`  
Visual Basic        `Sub Spread_KeyPress([Index As Integer,] KeyAscii As Integer)`

## OCX Use

The KeyPress event is a stock event.

## Parameters

The following parameters are available:

Parameter	Description
<i>Index</i>	Identifies a control if it is in an array
<i>KeyAscii</i>	Returns a standard numeric ANSI keycode ( <i>KeyAscii</i> is passed by reference; changing it sends a different character to the object. Changing <i>KeyAscii</i> to 0 cancels the keystroke so that the object does not receive a character.)

## Remarks

The object with the focus receives the event. A KeyPress event can involve any printable keyboard character, the Ctrl key combined with a character from the standard alphabet or one of a few special characters, and the Enter or Backspace key.

Use the [KeyDown](#) and [KeyUp](#) events to handle any keystroke not recognized by the KeyPress event, such as function keys, editing keys, navigation keys, or any combinations of these with the Alt, Ctrl, or Shift keys. KeyPress does not indicate the physical state of the keyboard; instead, it passes a character.

Visual Basic users can refer to the Visual Basic documentation for additional information about this event.

## LeaveCell Event

### Description

Occurs when the user moves the focus from one cell to another or moves the focus outside of the spreadsheet.

### Syntax

C++ (MFC)	<code>afx_msg void OnLeaveCellSpread(UINT, int, CWnd*, LPVOID);</code>
C++ (OWL)	<code>void EvLeaveCell(VBXEVENT far* event);</code>
Visual Basic (VBX)	<code>Sub Spread_LeaveCell(Col As Long, Row As Long, NewCol As Long, —NewRow As Long, Cancel As Integer)</code>
Visual Basic (OCX)	<code>Sub Spread_LeaveCell(ByVal Col As Long, ByVal Row As Long, —ByVal NewCol As Long, ByVal NewRow As Long,</code>

`—Cancel As Boolean`)

### Parameters

The following parameters are available:

Parameter	Description
<i>Col</i>	Column number of cell losing the focus
<i>Row</i>	Row number of cell losing the focus
<i>NewCol</i>	Column number of cell receiving the focus
<i>NewRow</i>	Row number of cell receiving the focus
<i>Cancel</i>	Set this value to True to prevent the focus from moving

### Remarks

The LeaveCell event occurs before the focus moves, letting you prevent certain actions.

If the focus is moved outside of the spreadsheet, both the *NewCol* and *NewRow* parameters will return —1.

## LeaveRow Event

[See Also](#)

### Description

Occurs when the operation mode is set to row mode and the focus moves to a new row.

### Syntax

C++ (MFC)           afx\_msg **void** OnLeaveRowSpread(**UINT**, **int**, **CWnd\***, **LPVOID**);  
C++ (OWL)           **void** EvLeaveRow(**VBXEVENT far\*** event);  
Visual Basic (VBX)   Sub *Spread\_LeaveRow*(Row As **Long**, RowWasLast As **Integer**,  
                          —RowChanged As **Integer**,

—AllCellsHaveData As **Integer**,

—NewRow As **Long**, NewRowsLast As **Integer**,

—Cancel As **Integer**)

Visual Basic (OCX)   Sub *Spread\_LeaveRow*(ByVal Row As **Long**,  
                          —ByVal RowWasLast As **Boolean**,

—ByVal RowChanged As **Boolean**,

—ByVal AllCellsHaveData As **Boolean**,

—ByVal NewRow As **Long**,

—ByVal NewRowsLast As **Long**, Cancel As **Boolean**)

### Parameters

The following parameters are available:

Parameter	Description
<i>Row</i>	Row number of row losing the focus
<i>RowWasLast</i>	Value is nonzero if row losing the focus is beyond the last row containing data (Treat this row as the new row.)
<i>RowChanged</i>	Value is nonzero if data within row losing the focus has changed
<i>AllCellsHaveData</i>	Value is nonzero if all cells within row losing the focus contain data
<i>NewRow</i>	Row number of row receiving the focus
<i>NewRowsLast</i>	Value is nonzero if row receiving the focus is beyond the last row containing data
<i>Cancel</i>	Set this value to True to prevent the focus from moving to the new row

### See Also

[OperationMode](#) property

## LostFocus Event

[See Also](#)

### Description

Occurs when an object loses the focus, either by a user action such as tabbing to or clicking another object, or by changing the focus in code using the SetFocus method.

### Syntax

C++ (MFC)	<code>afx_msg void OnLostFocusSpread(<b>UINT</b>, <b>int</b>, <b>CWnd*</b>, <b>LPVOID</b>);</code>
C++ (OWL)	<code>void EvLostFocus(<b>VBXEVENT far*</b> event);</code>
Visual Basic	<code>Sub Spread_LostFocus([<i>Index</i> As <b>Integer</b>])</code>

### OCX Use

The LostFocus event is a standard extender event that can be provided by your container.

### Parameter

The following parameter is available:

Parameter	Description
<i>Index</i>	Identifies a control if it is in an array

### Remarks

Use the LostFocus event primarily for verification and validation updates. Using the LostFocus event can cause validation to take place as the user leaves the control. You can use this event for enabling, disabling, hiding, and displaying other objects. You can also reverse or change conditions you set up in the object's [GotFocus](#) event.

Visual Basic users can refer to the Visual Basic documentation for additional information about this event.

**See Also**

[TabIndex](#), [TabStop](#) properties

[GotFocus](#) event



## MouseDown, MouseUp Events

[See Also](#)

### Description

Occur when the user presses (MouseDown) or releases (MouseUp) a mouse button.

### Syntax

C++ (MFC)	<code>afx_msg void OnMouseDownSpread(UINT, int, CWnd*, LPVOID);</code> <code>afx_msg void OnMouseUpSpread(UINT, int, CWnd*, LPVOID);</code>
C++ (OWL)	<code>void EvMouseDown(VBXEVENT far* event);</code> <code>void EvMouseUp(VBXEVENT far* event);</code>
Visual Basic	<code>Sub Spread_MouseDown([Index As Integer,] Button As Integer,</code> <code>—Shift As Integer, X As Single, Y As Single)</code>  <code>Sub Spread_MouseUp([Index As Integer,] Button As Integer,</code> <code>—Shift As Integer, X As Single, Y As Single)</code>

### OCX Use

The MouseDown and MouseUp events are stock events.

### Parameters

The following parameters are available:

Parameter	Description
<i>Index</i>	Identifies a control if it is in an array
<i>Button</i>	The button that was pressed (MouseDown) or released (MouseUp) to cause the event (The <i>Button</i> parameter is a bit field with bits corresponding to the left button (bit 0), the right button (bit 1), and the middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set, indicating which button caused the event.)
<i>Shift</i>	The state of the Alt, Ctrl, and Shift keys when the button specified in the <i>Button</i> parameter was pressed or released (A bit is set if the key is down. The <i>Shift</i> parameter is a bit field, with the least-significant bits corresponding to the Shift key (bit 0), the Ctrl key (bit 1), and the Alt key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both Ctrl and Alt are pressed, the value of <i>Shift</i> is 6.)
<i>X, Y</i>	The current horizontal (X) and vertical (Y) position of the mouse pointer (These coordinates are always expressed in terms of the target control's coordinate system.)

### Remarks

Use a MouseDown or MouseUp event to specify actions to occur when a given mouse button is pressed or released. Unlike the [Click](#) event, the MouseDown and MouseUp events allow you to distinguish between the left, right, and middle mouse buttons.

You can also write code for mouse-keyboard combinations that use the Alt, Ctrl, and Shift keys.

**Note** You can use a [MouseMove](#) event to respond to an event caused by moving the mouse. The *Button* parameter for the MouseDown and MouseUp events differs from the *Button* parameter used for the MouseMove event. For the MouseDown and MouseUp events, the *Button* parameter indicates one button per event; for the MouseMove event, it indicates the current state of all buttons.

Visual Basic users can refer to the Visual Basic documentation for additional information about this event.

**See Also**

[Click](#), [MouseMove](#) events

## MouseMove Event

[See Also](#)

### Description

Occurs when the user moves the mouse.

### Syntax

C++ (MFC)                `afx_msg void OnMouseMoveSpread(UINT, int, CWnd*, LPVOID);`  
C++ (OWL)                `void EvMouseMove(VBXEVENT far* event);`  
Visual Basic              `Sub Spread_MouseMove([Index As Integer,] Button As Integer,  
— Shift As Integer, X As Single, Y As Single)`

### OCX Use

The MouseMove event is a stock event.

### Parameters

The following parameters are available:

Parameter	Description
<i>Index</i>	Identifies a control if it is in an array
<i>Button</i>	The state of the mouse buttons, in which a bit is set if the button is down (The <i>Button</i> parameter is a bit field with bits corresponding to the left button (bit 0), the right button (bit 1), and the middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. This parameter indicates the complete state of the mouse buttons. Some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons are pressed.)
<i>Shift</i>	The state of the Alt, Ctrl, and Shift keys (A bit is set if the key is down. The <i>Shift</i> parameter is a bit field, with the least-significant bits corresponding to the Shift key (bit 0), the Ctrl key (bit 1), and the Alt key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both Ctrl and Alt are pressed, the value of <i>Shift</i> is 6.)
<i>X, Y</i>	The current horizontal ( <i>X</i> ) and vertical ( <i>Y</i> ) position of the mouse pointer (These coordinates are always expressed in terms of the target control's coordinate system.)

### Remarks

The MouseMove event is generated continually as the mouse pointer moves across objects. Unless another object has captured the mouse, an object recognizes a MouseMove event whenever the mouse position is within its borders.

You can use the [MouseDown](#) and [MouseUp](#) events to respond to events caused by pressing and releasing mouse buttons.

**Note** The *Button* parameter for the MouseDown and MouseUp events differs from the *Button* parameter used for the MouseMove event. For the MouseMove event, the *Button* parameter indicates the current state of all buttons; for the MouseDown and MouseUp events, it indicates one button per event.

Visual Basic users can refer to the Visual Basic documentation for additional information about this event.

**See Also**

[Click](#), [MouseDown](#), [MouseUp](#) events

## PrintAbort Event

[See Also](#)

### Description

Occurs repeatedly when the spreadsheet is printing.

### Syntax

C++ (MFC)	<code>afx_msg void OnPrintAbortSpread(<i>UINT, int, CWnd*, LPVOID</i>);</code>
C++ (OWL)	<code>void EvPrintAbort(<i>VBXEVENT far*</i> event);</code>
Visual Basic (VBX)	<code>Sub Spread_PrintAbort(<i>Abort As Integer</i>)</code>
Visual Basic (OCX)	<code>Sub Spread_PrintAbort(<i>Abort As Boolean</i>)</code>

### Parameter

The following parameter is available:

Parameter	Description
<i>Abort</i>	Status of print job

### Remarks

When the user clicks the Cancel button in a print in-progress dialog box, the Spread control passes the *Abort* parameter as True to inform the application that printing has stopped.

The application can set the *Abort* parameter at any time to stop printing.

**See Also**

[Action](#) (13 - Print, 32 - Smart Print) property

## QueryAdvance Event

### Description

Occurs when the user moves the focus to the next or previous control on the form. This event occurs only in Visual Basic version 2.0 or later.

### Syntax

Visual Basic (VBX)     Sub *Spread\_QueryAdvance*(*AdvanceNext* As **Integer**,  
                          — *Cancel* As **Integer**)

Visual Basic (OCX)    Sub *Spread\_QueryAdvance*(ByVal *AdvanceNext* As **Boolean**,  
                          — *Cancel* As **Boolean**)

### Parameters

The following parameters are available:

Parameter	Description
<i>AdvanceNext</i>	True if the user pressed the Tab or Down Arrow key; False if the user pressed Shift+Tab or the Up Arrow key
<i>Cancel</i>	Defaults to True and the focus stays within the Spread control (Set this value to False to let the focus move to the next or previous control on the form. If True, the user cannot move the focus.)

### Remarks

This event occurs when the focus is in

- the upper-left cell and the user presses Shift+Tab
- the lower-right cell and the user presses the Tab key
- the top row and the user presses the Up Arrow key
- the bottom row and the user presses the Down Arrow key

The application responds by moving the focus to the next or previous control on the form.

## QueryData Event

[See Also](#)

### Description

Occurs when virtual mode is on and the Spread control requests additional data.

### Syntax

C++ (MFC)                   afx\_msg **void** OnQueryDataSpread(**UINT**, **int**, **CWnd\***, **LPVOID**);

C++ (OWL)                   **void** EvQueryData(**VBXEVENT far\*** event);

Visual Basic (VBX)       Sub *Spread\_QueryData*(*Row* As **Long**, *RowsNeeded* As **Long**,  
                          —*RowsLoaded* As **Long**, *Direction* As **Integer**,

—*AtTop* As **Integer**, *AtBottom* As **Integer**)

Visual Basic (OCX)       Sub *Spread\_QueryData*(ByVal *Row* As **Long**,  
                          —ByVal *RowsNeeded* As **Long**,

—*RowsLoaded* As **Long**,

—ByVal *Direction* As **Integer**,

—*AtTop* As **Boolean**, *AtBottom* As **Boolean**)

### Parameters

The following parameters are available:

Parameter	Description										
<i>Row</i>	Row number of first row requesting data										
<i>RowsNeeded</i>	Number of rows to fill with data										
<i>RowsLoaded</i>	Number of rows loaded										
<i>Direction</i>	One of the following values represents the direction the user was moving in the Spread control that caused this event to occur: <table><thead><tr><th>Value</th><th>Description</th></tr></thead><tbody><tr><td>1</td><td>Down</td></tr><tr><td>2</td><td>Up</td></tr><tr><td>3</td><td>Top</td></tr><tr><td>4</td><td>Bottom</td></tr></tbody></table>	Value	Description	1	Down	2	Up	3	Top	4	Bottom
Value	Description										
1	Down										
2	Up										
3	Top										
4	Bottom										
<i>AtTop</i>	Set to True if the top of the data is reached										
<i>AtBottom</i>	Set to True if the bottom of the data is reached										

### Remarks

*Direction* parameter values 3 (Top) and 4 (Bottom) are returned only if the [VScrollSpecial](#) property is set to True.



**See Also**

[VirtualMode](#), [VirtualRows](#), [VScrollSpecial](#), [VScrollSpecialType](#) properties

## RightClick Event

[See Also](#)

### Description

Occurs when the user clicks a cell with the right mouse button.

### Syntax

C++ (MFC)                    `afx_msg void OnRightClickSpread(UINT, int, CWnd*, LPVOID);`

C++ (OWL)                    `void EvRightClick(VBXEVENT far* event);`

Visual Basic (VBX)        `Sub Spread_RightClick(ClickType As Integer, Col As Long,  
—Row As Long, MouseX As Long,`

`—MouseY As Long)`

Visual Basic (OCX)        `Sub Spread_RightClick(ByVal ClickType As Integer,  
—ByVal Col As Long, ByVal Row As Long,`

`—ByVal MouseX As Long, ByVal MouseY As Long)`

### Parameters

The following parameters are available:

Parameter	Description								
<i>ClickType</i>	Can be one of the following values: <table><thead><tr><th>Value</th><th>Description</th></tr></thead><tbody><tr><td>0</td><td>Down</td></tr><tr><td>1</td><td>Up</td></tr><tr><td>2</td><td>Double-click</td></tr></tbody></table>	Value	Description	0	Down	1	Up	2	Double-click
Value	Description								
0	Down								
1	Up								
2	Double-click								
<i>Col</i>	Column number of cell the user clicked								
<i>Row</i>	Row number of cell the user clicked								
<i>MouseX</i>	X position of mouse pointer relative to upper-left corner of spreadsheet (This value is in twips.)								
<i>MouseY</i>	Y position of mouse pointer relative to upper-left corner of spreadsheet (This value is in twips.)								

### See Also

[Click](#), [DbClick](#) events

## RowHeightChange Event

[See Also](#)

### Description

Occurs when the user changes the height of a row.

### Syntax

C++ (MFC)	<code>afx_msg void OnRowHeightSpread(UINT, int, CWnd*, LPVOID);</code>
C++ (OWL)	<code>void EvRowHeight(VBXEVENT far* event);</code>
Visual Basic (VBX)	<code>Sub Spread_RowHeightChange(Row1 As Long, Row2 As Long)</code>
Visual Basic (OCX)	<code>Sub Spread_RowHeightChange(ByVal Row1 As Long, —ByVal Row2 As Long)</code>

### Parameters

The following parameters are available:

Parameter	Description
<i>Row1</i>	Row number of first row whose height changed
<i>Row2</i>	Row number of last row whose height changed

### Remarks

In most cases, the row value for the *Row2* parameter is the same as the value of the *Row1* parameter. If the value is larger, the heights of the range of rows specified by *Row1* and *Row2* have all changed.

**See Also**

[RowHeight](#), [UserResize](#), [UserResizeRow](#) properties

[ColWidthChange](#) event

## SelChange Event

[See Also](#)

### Description

Occurs while the user is selecting a block of cells.

### Syntax

C++ (MFC)                    `afx_msg void OnSelChangeSpread(UINT, int, CWnd*, LPVOID);`

C++ (OWL)                    `void EvSelChange(VBXEVENT far* event);`

Visual Basic (VBX)        `Sub Spread_SelChange(BlockCol As Long, BlockRow As Long,  
—BlockCol2 As Long, BlockRow2 As Long,`

`—CurCol As Long, CurRow As Long)`

Visual Basic (OCX)        `Sub Spread_SelChange(ByVal BlockCol As Long,  
—ByVal BlockRow As Long,`

`—ByVal BlockCol2 As Long,`

`—ByVal BlockRow2 As Long,`

`—ByVal CurCol As Long,`

`—ByVal CurRow As Long)`

### Parameters

The following parameters are available:

Parameter	Description
<i>BlockCol</i>	Column number of cell in upper-left corner of block
<i>BlockRow</i>	Row number of cell in upper-left corner of block
<i>BlockCol2</i>	Column number of cell in lower-right corner of block
<i>BlockRow2</i>	Row number of cell in lower-right corner of block
<i>CurCol</i>	Column number of cell containing mouse pointer
<i>CurRow</i>	Row number of cell containing mouse pointer

### Remarks

The SelChange event occurs each time the selected block changes when the user extends the selection by dragging the mouse or by pressing the Shift key and an arrow key simultaneously.

**See Also**

[SelectBlockOptions](#) property

[BlockSelected](#) event

## TopLeftChange Event

[See Also](#)

### Description

Occurs when the left column and/or top row displayed changes when the user scrolls using the keyboard or the scroll bars.

### Syntax

C++ (MFC)                   afx\_msg **void** OnTopLeftChangeSpread(**UINT**, **int**, **CWnd\***, **LPVOID**);

C++ (OWL)                   **void** EvTopLeftChange(**VBXEVENT far\*** event);

Visual Basic (VBX)       Sub *Spread\_TopLeftChange*(*OldLeft* As **Long**, *OldTop* As **Long**,  
—*NewLeft* As **Long**, *NewTop* As **Long**)

Visual Basic (OCX)       Sub *Spread\_TopLeftChange*(ByVal *OldLeft* As **Long**,  
—ByVal *OldTop* As **Long**,

—ByVal *NewLeft* As **Long**, ByVal *NewTop* As **Long**)

### Parameters

The following parameters are available:

<b>Parameter</b>	<b>Description</b>
<i>OldLeft</i>	Column number of previous left column
<i>OldTop</i>	Row number of previous top row
<i>NewLeft</i>	Column number of new left column (If the value of <i>OldLeft</i> equals the value of <i>NewLeft</i> , the spreadsheet did not scroll horizontally.)
<i>NewTop</i>	Row number of new top row (If the value of <i>OldTop</i> equals the value of <i>NewTop</i> , the spreadsheet did not scroll vertically.)

### See Also

[LeftCol](#), [TopRow](#) properties

## UserFormulaEntered Event

[See Also](#)

### Description

Occurs when the user types a formula.

### Syntax

C++ (MFC)	<code>afx_msg void OnUserFormulaEnteredSpread(UINT, int, CWnd*, LPVOID);</code>
C++ (OWL)	<code>void EvUserFormulaEntered(VBXEVENT far* event);</code>
Visual Basic (VBX)	<code>Sub Spread_UserFormulaEntered(Col As Long, Row As Long)</code>
Visual Basic (OCX)	<code>Sub Spread_UserFormulaEntered(ByVal Col As Long, ByVal Row As Long)</code>

### Parameters

The following parameters are available:

Parameter	Description
<i>Col</i>	Column number of cell containing formula
<i>Row</i>	Row number of cell containing formula

### Remarks

The application can access the formula using the [Formula](#) property.



**See Also**

[AllowUserFormulas](#), [Formula](#) properties

## VirtualClearData Event

[See Also](#)

### Description

Occurs when the spreadsheet is in virtual mode and data in the virtual buffer needs to be discarded.

### Syntax

C++ (MFC)	<code>afx_msg void OnVirtualClearDataSpread(UINT, int, CWnd*, —LPVOID);</code>
C++ (OWL)	<code>void EvVirtualClearData(VBXEVENT far* event);</code>
Visual Basic (VBX)	<code>Sub Spread_VirtualClearData(Row As Long, —RowsBeingCleared As Long)</code>
Visual Basic (OCX)	<code>Sub Spread_VirtualClearData(ByVal Row As Long, —ByVal RowsBeingCleared As Long)</code>

### Parameters

The following parameters are available:

Parameter	Description
<i>Row</i>	Row number of top or first row to discard
<i>RowsBeingCleared</i>	Number of rows to discard

### Remarks

The VirtualClearData event occurs before the data in the virtual buffer is cleared, giving the application an opportunity to process the data before it is discarded.

**See Also**

[VirtualMode](#), [VirtualRows](#), [VScrollSpecial](#), [VScrollSpecialType](#) properties

[QueryData](#) event



[GetClientArea](#)  
[GetCollItemData](#)  
[GetCustomName](#)  
[GetDataFillData](#)  
[GetFirstValidCell](#)  
[GetFormulaSync](#)  
[GetItemData](#)  
[GetIteration](#)  
[GetLastValidCell](#)  
[GetMultiSellItem](#)  
[GetRefStyle](#)  
[GetRowItemData](#)  
[GetText](#)

## I

[IsCellSelected](#)  
[IsFormulaValid](#)  
[IsVisible](#)

## L

[LoadFromFile](#)  
[LoadTabFile](#)

## Q

[QueryCustomName](#)

## R

[RowHeightToTwips](#)

## S

[SaveTabFile](#)  
[SaveToFile](#)  
[SetCellDirtyFlag](#)  
[SetCollItemData](#)  
[SetCustomName](#)  
[SetDataFillData](#)  
[SetFormulaSync](#)  
[SetItemData](#)  
[SetIteration](#)  
[SetRefStyle](#)  
[SetRowItemData](#)  
[SetText](#)

## T

[TwipsToColWidth](#)  
[TwipsToRowHeight](#)

No functions begin with this letter.

## AddCustomFunction Function

[See Also](#)    [Example](#)

### Description

Defines a custom function, which can be used in a formula definition.

### Syntax

C++ (OCX)	<b>BOOL</b> CSpreadSheet::AddCustomFunction ( <b>LPCTSTR</b> <i>FunctionName</i> , — <b>short</b> <i>ParameterCnt</i> );
C++ (VBX)	<b>BOOL</b> CSpreadSheet::AddCustomFunction ( <b>LPCSTR</b> <i>FunctionName</i> , — <b>short</b> <i>ParameterCnt</i> );
Visual Basic (OCX)	Function <i>Spread</i> .AddCustomFunction (ByVal <i>FunctionName</i> As <b>String</b> , —ByVal <i>ParameterCnt</i> As <b>Integer</b> ) As <b>Boolean</b>
Visual Basic (VBX)	Function SpreadAddCustomFunction (SS As <b>Control</b> , —ByVal <i>FunctionName</i> As <b>String</b> ,

—ByVal *ParameterCnt* As **Integer**) As **Integer**

### Parameters

The following parameters are available:

Parameter	Description
SS	Spread control name
<i>FunctionName</i>	Name of the custom function
<i>ParameterCnt</i>	Number of parameters the function expects (Set this value to —1 if any number of parameters are allowed.)

### Remarks

An application must respond to the [CustomFunction](#) event to implement the custom function.

### Return Type

VBX: Nonzero if successful; otherwise, 0.  
OCX: TRUE if successful; otherwise, FALSE.

[Print](#)

[Copy](#)

[Close](#)

The following example creates three custom functions.

C++

```
ret = vaSpread1->AddCustomFunction("SIN", 1);  
ret = vaSpread1->AddCustomFunction("COS", 1);  
ret = vaSpread1->AddCustomFunction("TAN", 1);
```

OCX

```
ret = vaSpread1.AddCustomFunction("SIN", 1)  
ret = vaSpread1.AddCustomFunction("COS", 1)  
ret = vaSpread1.AddCustomFunction("TAN", 1)
```

Visual Basic

```
ret = SpreadAddCustomFunction(SS, "SIN", 1)  
ret = SpreadAddCustomFunction(SS, "COS", 1)  
ret = SpreadAddCustomFunction(SS, "TAN", 1)
```



**See Also**

[CustomFunction](#) event

[AddCustomFunctionExt](#), [CFGetCellParam](#), [CFGetDoubleParam](#), [CFGetDoubleParamExt](#), [CFGetLongParam](#), [CFGetParamInfo](#), [CFGetRangeParam](#), [CFGetStringParam](#), [CFSetResult](#), [DerefHlstrLen](#) functions

## AddCustomFunctionExt Function

[See Also](#)

### Description

Defines a custom function with a variable number of parameters.

### Syntax

C++ (OCX)                    **BOOL** CSpreadSheet::AddCustomFunctionExt  
                                  —(**LPCWSTR** *FunctionName*, **short** *MinParamCnt*,  
  
—**short** *MaxParamCnt*, **long** *Flags*);  
C++ (VBX)                    **BOOL** CSpreadSheet::AddCustomFunctionExt  
                                  —(**LPCSTR** *FunctionName*, **short** *MinParamCnt*,  
  
—**short** *MaxParamCnt*, **long** *Flags*);  
Visual Basic (OCX)        Function *Spread*.AddCustomFunctionExt  
                                  —(ByVal *FunctionName* As **String**,  
  
—ByVal *MinParamCnt* As **Integer**,  
  
—ByVal *MaxParamCnt* As **Integer**,  
  
—ByVal *Flags* As **Long**) As **Boolean**  
Visual Basic (VBX)        Function *Spread*.AddCustomFunctionExt (*SS* As **Control**,  
                                  —ByVal *FunctionName* As **String**,  
  
—ByVal *MinParamCnt* As **Integer**,  
  
—ByVal *MaxParamCnt* As **Integer**,  
  
—ByVal *Flags* As **Long**) As **Integer**

### Parameters

The following parameters are available:

Parameter	Description									
<i>SS</i>	Spread control name									
<i>FunctionName</i>	Name of the custom function									
<i>MinParamCnt</i>	Minimum number of parameters required by the function									
<i>MaxParamCnt</i>	Maximum number of parameters required by the function									
<i>Flags</i>	Requests references instead of cell values Combine either of the following values with the Or operator:									
	<table><thead><tr><th>Return Value</th><th>Constant</th><th>Description</th></tr></thead><tbody><tr><td>1</td><td>SS_CUSTFUNC_WANTCELLREF</td><td>Function receives cell references</td></tr><tr><td>2</td><td>SS_CUSTFUNC_WANTRANGEREFF</td><td>Function receives range references</td></tr></tbody></table>	Return Value	Constant	Description	1	SS_CUSTFUNC_WANTCELLREF	Function receives cell references	2	SS_CUSTFUNC_WANTRANGEREFF	Function receives range references
Return Value	Constant	Description								
1	SS_CUSTFUNC_WANTCELLREF	Function receives cell references								
2	SS_CUSTFUNC_WANTRANGEREFF	Function receives range references								

### Remarks

An application must respond to the [CustomFunction](#) event to implement the custom function.

### Return Type

VBX: Nonzero if successful; otherwise, 0.  
OCX: TRUE if successful; otherwise, FALSE.

**See Also**

[CustomFunction](#) event

[AddCustomFunction](#), [CFGetCellParam](#), [CFGetDoubleParam](#), [CFGetDoubleParamExt](#), [CFGetLongParam](#), [CFGetParamInfo](#), [CFGetRangeParam](#), [CFGetStringParam](#), [CFSetResult](#), [DerefHlstrLen](#) functions

## CFGetCellParam Function

[See Also](#)

### Description

Returns the column and row coordinates of a cell reference parameter that is being passed into a custom function.

### Syntax

C++ (OCX)	<code>void CSpreadSheet::CFGetCellParam (<i>short Param</i>, —<i>long* Col</i>, <i>long* Row</i>);</code>
C++ (VBX)	<code>void CSpreadSheet::CFGetCellParam (<i>short Param</i>, —<i>long FAR* Col</i>, <i>long FAR* Row</i>);</code>
Visual Basic (OCX)	Function <i>Spread</i> .CFGetCellParam (ByVal <i>Param</i> As <i>Integer</i> , — <i>Col</i> As <i>Long</i> , <i>Row</i> As <i>Long</i> )
Visual Basic (VBX)	Sub SpreadCFGetCellParam (SS As <i>Control</i> , —ByVal <i>Param</i> As <i>Integer</i> , <i>Col</i> As <i>Long</i> ,

—*Row* As *Long*)

### Parameters

The following parameters are available:

Parameter	Description
<i>SS</i>	Spread control name
<i>Param</i>	Index number of the parameter (The index of the first parameter is 1.)
<i>Col</i>	Column coordinate of cell reference
<i>Row</i>	Row coordinate of cell reference

### Remarks

This function can only be used inside the [CustomFunction](#) event handler.

You must use the [CFGetParamInfo](#) function to check the parameter status and type before calling the CFGetCellParam function.

### Return Type

None

**See Also**

[CustomFunction](#) event

[AddCustomFunction](#), [AddCustomFunctionExt](#), [CFGetDoubleParam](#), [CFGetDoubleParamExt](#), [CFGetLongParam](#), [CFGetParamInfo](#), [CFGetRangeParam](#), [CFGetStringParam](#), [CFSetResult](#), [DerefHlstrLen](#) functions

## CFGetDoubleParam Function

[See Also](#)      [Example](#)

### Description

Returns a custom function's parameter as a double-precision, floating-point value.

### Syntax

C++	<code><b>double</b> CSpreadSheet::CFGetDoubleParam (<i>short Param</i>);</code>
Visual Basic (OCX)	Function <i>Spread</i> .CFGetDoubleParam (ByVal <i>Param</i> As <b>Integer</b> ) —As <b>Double</b>
Visual Basic (VBX)	Function SpreadCFGetDoubleParam (SS As <b>Control</b> , —ByVal <i>Param</i> As <b>Integer</b> ) As <b>Double</b>

### Parameters

The following parameters are available:

Parameter	Description
SS	Spread control name
<i>Param</i>	Index number of the parameter (The index of the first parameter is 1.)

### Remarks

Borland users must use the [CFGetDoubleParamExt](#) function. This function can only be used inside the [CustomFunction](#) event handler.

You must use the [CFGetParamInfo](#) function to check the parameter status and type before calling the CFGetDoubleParam function.

### Return Type

Value of the specified parameter

[Print](#)

[Copy](#)

[Close](#)

The following example returns the first and second parameters of a custom function as double-precision, floating-point values.

C++

```
Val1 = vaSpread1->CFGetDoubleParam(1);  
Val2 = vaSpread1->CFGetDoubleParam(2);
```

OCX

```
Val1 = vaSpread1.CFGetDoubleParam(1)  
Val2 = vaSpread1.CFGetDoubleParam(2)
```

Visual Basic

```
Val1 = SpreadCFGetDoubleParam(SS, 1)  
Val2 = SpreadCFGetDoubleParam(SS, 2)
```

**See Also**

[CustomFunction](#) event

[AddCustomFunction](#), [AddCustomFunctionExt](#), [CFGetCellParam](#), [CFGetDoubleParamExt](#), [CFGetLongParam](#), [CFGetParamInfo](#), [CFGetRangeParam](#), [CFGetStringParam](#), [CFSetResult](#), [DerefHlstrLen](#) functions



## CFGetDoubleParamExt Function

[See Also](#)

### Description

Returns a custom function's parameter as a double-precision, floating-point value. Borland C++ users must use this function instead of the [CFGetDoubleParam](#) function.

### Syntax

C++ (OCX)	<code>void CSpreadSheet::CFGetDoubleParamExt (<b>short</b> Param, —<b>double FAR*</b> ParamValue);</code>
C++ (VBX)	<code>void CSpreadSheet::CFGetDoubleParamExt (<b>short</b> Param, —<b>double FAR*</b> lpdfValue);</code>
Visual Basic (OCX)	Function <i>Spread</i> .CFGetDoubleParamExt (ByVal <i>Param</i> As <b>Integer</b> , — <i>ParamValue</i> As <b>Double</b> )
Visual Basic (VBX)	Function <i>Spread</i> CFGetDoubleParamExt (SS As <b>Control</b> , —ByVal <i>Param</i> As <b>Integer</b> , <i>ParamValue</i> As <b>Double</b> )

### Parameters

The following parameters are available:

#### OCX, Visual Basic (VBX) Parameters

Parameter	Description
SS	Spread control name (VB only)
<i>Param</i>	Index number of the parameter (The index of the first parameter is 1.)
<i>ParamValue</i>	Value of the specified parameter

#### C++ (VBX) Parameters

Parameter	Description
<i>Param</i>	Index number of the parameter (The index of the first parameter is 1.)
<i>lpdfValue</i>	Value of the specified parameter

### Remarks

This function can only be used inside the [CustomFunction](#) event handler.

You must use the [CFGetParamInfo](#) function to check the parameter status and type before calling the CFGetDoubleParamExt function.

### Return Type

Value of the specified parameter

**See Also**

[CustomFunction](#) event

[AddCustomFunction](#), [AddCustomFunctionExt](#), [CFGetCellParam](#), [CFGetDoubleParam](#), [CFGetLongParam](#), [CFGetParamInfo](#), [CFGetRangeParam](#), [CFGetStringParam](#), [CFSetResult](#), [DerefHlstrLen](#) functions

## CFGetLongParam Function

[See Also](#)    [Example](#)

### Description

Returns a custom function's parameter as a long integer value.

### Syntax

C++	<b>long</b> CSpreadSheet::CFGetLongParam ( <b>short</b> <i>Param</i> );
Visual Basic (OCX)	Function <i>Spread</i> .CFGetLongParam (ByVal <i>Param</i> As <b>Integer</b> ) —As <b>Long</b>
Visual Basic (VBX)	Function <i>Spread</i> .CFGetLongParam ( <i>SS</i> As <b>Control</b> , —ByVal <i>Param</i> As <b>Integer</b> ) As <b>Long</b>

### Parameters

The following parameters are available:

Parameter	Description
<i>SS</i>	Spread control name
<i>Param</i>	Index number of the parameter (The index of the first parameter is 1.)

### Remarks

This function can only be used inside the [CustomFunction](#) event handler.

You must use the [CFGetParamInfo](#) function to check the parameter status and type before calling the CFGetLongParam function.

### Return Type

Value of the specified parameter

[Print](#)

[Copy](#)

[Close](#)

The following example returns the first and second parameters of a custom function as long integer values.

C++

```
Val1 = vaSpread1->CFGetLongParam(1);  
Val2 = vaSpread1->CFGetLongParam(2);
```

OCX

```
Val1 = vaSpread1.CFGetLongParam(1)  
Val2 = vaSpread1.CFGetLongParam(2)
```

Visual Basic

```
Val1 = SpreadCFGetLongParam(SS, 1)  
Val2 = SpreadCFGetLongParam(SS, 2)
```

**See Also**

[CustomFunction](#) event

[AddCustomFunction](#), [AddCustomFunctionExt](#), [CFGetCellParam](#), [CFGetDoubleParam](#), [CFGetDoubleParamExt](#), [CFGetParamInfo](#), [CFGetRangeParam](#), [CFGetStringParam](#), [CFSetResult](#), [DerefHlstrLen](#) functions



[Print](#)

[Copy](#)

[Close](#)

The following example retrieves information about the first parameter of a custom function.

C++

```
Cstring sval;
ret = vaSpread1->CFGetParamInfo(1, &wType, &wStatus);
if((ret==FALSE)|| (wStatus != SS_VALUE_STATUS_OK))
{
    MessageBox(NULL, "Parameter Error", "Notice", MB_OK);
    return;
}
if(wType==SS_VALUE_TYPE_STR)
    sval = vaSpread1->CFGetStringParam(1);
else if(wType==SS_VALUE_TYPE_LONG)
    lVal = vaSpread1->CFGetLongParam(1);
else if(wType==SS_VALUE_TYPE_DOUBLE)
    dfVal = vaSpread1->CFGetDoubleParam(1);
```

OCX

```
ret = vaSpread1.CFGetParamInfo(1, wType, wStatus )
If (ret = False) OR (wStatus != SS_VALUE_STATUS_OK) Then
MsgBox "Parameter Error", 48, "Notice"
Exit Sub
End If
If wType = SS_VALUE_TYPE_STR Then
Val = vaSpread1.CFGetStringParam(1)
Else If wType = SS_VALUE_TYPE_LONG Then
Val = vaSpread1.CFGetLongParam(1)
Else If wType = SS_VALUE_TYPE_DOUBLE
Val = vaSpread1.CFGetDoubleParam(1)
End If
```

Visual Basic

```
ret = SpreadCFGetParamInfo(SS, 1, wType, wStatus )
If (ret = False) OR (wStatus != SS_VALUE_STATUS_OK) Then
MsgBox "Parameter Error", 48, "Notice"
Exit Sub
End If
If wType = SS_VALUE_TYPE_STR Then
Val = SpreadCFGetStringParam(SS, 1)
Else If wType = SS_VALUE_TYPE_LONG Then
Val = SpreadCFGetLongParam(SS, 1)
Else If wType = SS_VALUE_TYPE_DOUBLE
Val = SpreadCFGetDoubleParam(SS, 1)
End If
```

**See Also**

[CustomFunction](#) event

[AddCustomFunction](#), [AddCustomFunctionExt](#), [CFGetCellParam](#), [CFGetDoubleParam](#), [CFGetDoubleParamExt](#), [CFGetLongParam](#), [CFGetRangeParam](#), [CFGetStringParam](#), [CFSetResult](#), [DerefHlstrLen](#) functions





## See Also

[CustomFunction](#) event

[AddCustomFunction](#), [AddCustomFunctionExt](#), [CFGetCellParam](#), [CFGetDoubleParam](#), [CFGetDoubleParamExt](#), [CFGetLongParam](#), [CFGetParamInfo](#), [CFGetStringParam](#), [CFSetResult](#), [DerefHlstrLen](#) functions



[Print](#)

[Copy](#)

[Close](#)

The following example retrieves a custom function's string parameters.

C++

```
CString Parm1, Parm2;  
Parm1 = vaSpread1->CFGetStringParam(1);  
Parm2 = vaSpread1->CFGetStringParam(2);
```

OCX

```
Parm1 = vaSpread1.CFGetStringParam(1)  
Parm2 = vaSpread1.CFGetStringParam(2)
```

Visual Basic

```
Parm1 = SpreadCFGetStringParam(SS, 1)  
Parm2 = SpreadCFGetStringParam(SS, 2)
```

## See Also

[CustomFunction](#) event

[AddCustomFunction](#), [AddCustomFunctionExt](#), [CFGetCellParam](#), [CFGetDoubleParam](#), [CFGetDoubleParamExt](#), [CFGetLongParam](#), [CFGetParamInfo](#), [CFGetRangeParam](#), [CFSetResult](#), [DereffHltrLen](#) functions

## CFSetResult Function

[See Also](#)    [Example](#)

### Description

Sets the return value of a custom function.

### Syntax

C++ (OCX)	<code>void CSpreadSheet::CFSetResult (<b>VARIANT*</b> Var);</code>
C++ (VBX)	<code>void CSpreadSheet::CFSetResult (<b>LPCSTR</b> lpszResult);</code> <code>void CSpreadSheet::CFSetResult (<b>long</b> lResult);</code> <code>void CSpreadSheet::CFSetResult (<b>double</b> dfResult);</code>
Visual Basic (OCX)	Function <i>Spread</i> .CFSetResult (Var As <b>VARIANT</b> )
Visual Basic (VBX)	Sub SpreadCFSetResult (SS As <b>Control</b> , Var As <b>VARIANT</b> )

### Parameters

The following parameters are available:

#### C++ (VBX) Parameters

Parameter	Description
<i>lpszResult</i>	Result returned (if string)
<i>lResult</i>	Result returned (if long)
<i>dfResult</i>	Result returned (if double)

#### OCX, VBX Parameters

Parameter	Description
SS	Spread control name
Var	Result returned

### Remarks

The application uses this function to return the result of the custom function.

### Return Type

None

[Print](#)

[Copy](#)

[Close](#)

The following example sets the return value of a custom function.

C++

```
vaSpread1->CFSetResult (ret) ;
```

OCX

```
vaSpread1.CFSetResult (ret)
```

Visual Basic

```
SpreadCFSetResult(SS, ret)
```

## See Also

[CustomFunction](#) event

[AddCustomFunction](#), [AddCustomFunctionExt](#), [CFGetCellParam](#), [CFGetDoubleParam](#), [CFGetDoubleParamExt](#), [CFGetLongParam](#), [CFGetParamInfo](#), [CFGetRangeParam](#), [CFGetStringParam](#), [DerefHlstrLen](#) functions



## ColNumberToLetter Function

[Example](#)

### Description

Converts a column number to the corresponding column letter.

### Syntax

C++	<b>CString</b> CSpreadSheet::ColNumberToLetter ( <b>long</b> HeaderNumber);
Visual Basic (OCX)	Function Spread.ColNumberToLetter (ByVal HeaderNumber As <b>Long</b> ) —As <b>String</b>
Visual Basic (VBX)	Function SpreadColNumberToLetter (SS As <b>Control</b> , —ByVal HeaderNumber As <b>Long</b> ) As <b>String</b>

### Parameters

The following parameter is available:

Parameter	Description
<i>HeaderNumber</i>	Column number of column being converted

### Return Type

String

[Print](#)

[Copy](#)

[Close](#)

The following example converts a column number to the corresponding column letter.

C++

```
Cstring s;  
s = vaSpread1->ColNumberToLetter(1Col);
```

Visual Basic

```
s$ = SpreadColNumberToLetter(55)
```

## ColWidthToTwips Function

[See Also](#)      [Example](#)

### Description

Converts a column-width measurement to twips.

### Syntax

C++ (OCX)	<code>void CSpreadSheet::ColWidthToTwips (<i>float</i> Width, <i>long*</i> Twips);</code>
C++ (VBX)	<code>void CSpreadSheet::ColWidthToTwips (<i>float</i> Width, —<i>long FAR*</i> Twips);</code>
Visual Basic (OCX)	Function <i>Spread.ColWidthToTwips</i> (ByVal <i>ColWidth</i> As <b>Single</b> , — <i>Twips</i> As <b>Long</b> )
Visual Basic (VBX)	Sub <i>SpreadColWidthToTwips</i> ( <i>SS</i> As <b>Control</b> , —ByVal <i>ColWidth</i> As <b>Single</b> , <i>Twips</i> As <b>Long</b> )

### Parameters

The following parameters are available:

#### C++ Parameters

Parameter	Description
<i>Width</i>	Width of column
<i>Twips</i>	Width of column returned in twips

#### Visual Basic Parameters

Parameter	Description
<i>SS</i>	Spread control name
<i>ColWidth</i>	Width of column
<i>Twips</i>	Width of column returned in twips

### Remarks

The value returned by the ColWidthToTwips function includes the width of the adjacent grid line.

### Return Type

None

[Print](#)

[Copy](#)

[Close](#)

The following example converts a column-width measurement to twips.

C++

```
vaSpread1->ColWidthToTwips (Width, &Twips);
```

OCX

```
vaSpread1.ColWidthToTwips (Width, Twips)
```

Visual Basic

```
SpreadColWidthToTwips (SS, Width, Twips)
```

**See Also**

[ColWidth](#), [RowHeight](#) properties

[RowHeightToTwips](#), [TwipsToColWidth](#), [TwipsToRowHeight](#) functions

## DerefHlstrLen Function

[See Also](#)      [Example](#)

### Description

Returns a pointer to the string data of a Basic language string and the length of the string. This function is for MFC VBX users only.

### Syntax

C++ (MFC VBX)      **LPSTR** CSpreadSheet::DerefHlstrLen (**HLSTR** hlstr, **short FAR\*** lpcbLen);

### Parameters

The following parameters are available:

Parameter	Description
<i>hlstr</i>	Handle to the Basic language string
<i>lpcbLen</i>	Length of the returned string data

### Remarks

Use this function to access Visual Basic strings which are passed as event parameters.

The returned pointer value becomes invalid as soon as the string is moved in memory; any call to a Visual Basic API function can have that effect. If this happens, the string must be dereferenced again to be a valid pointer.

### Return Type

A far pointer to the string data or NULL if the string is zero length. If the string has a length greater than zero, it is not null-terminated and can contain embedded nulls.

[Print](#)

[Copy](#)

[Close](#)

The following example creates a "cube(x)" function.

C++ (MFC VBX)

```
afx_msg void CMyWnd::OnCustomFunction(UINT, int, CWnd* pCtl, LPVOID lpParams)
{
    HLSTR hlstrFuncName = SS_CUSTOMFUNCTION_FUNCTIONNAME(lpParams);
    short nParamCnt = SS_CUSTOMFUNCTION_PARAMETERCNT(lpParams);
    long lCol = SS_CUSTOMFUNCTION_COL(lpParams);
    long lRow = SS_CUSTOMFUNCTION_ROW(lpParams);
    short nStatus = SS_VALUE_STATUS_EMPTY;
    short nFuncNameLen;
    LPSTR lpszFuncName = m_Spread.DerefHlstrLen(hlstrFuncName, &nFuncNameLen);
    if( _fstrnicmp("cube", lpszFuncName, nFuncNameLen) == 0 )
    {
        short nArgType;
        short nArgStatus;
        m_Spread.CFGetParamInfo(1, &nArgType, &nArgStatus);
        if( SS_VALUE_STATUS_OK == nArgStatus )
        {
            if( SS_VALUE_TYPE_LONG == nArgType )
            {
                long lArg = m_Spread.CFGetLongParam(1);
                long lResult = lArg * lArg * lArg;
                m_Spread.CFSetResult(lResult);
                nStatus = SS_VALUE_STATUS_OK;
            }
            else if( SS_VALUE_TYPE_DOUBLE == nArgType )
            {
                double dfArg = m_Spread.CFGetDoubleParam(1);
                double dfResult = dfArg * dfArg * dfArg;
                m_Spread.CFSetResult(dfResult);
                nStatus = SS_VALUE_STATUS_OK;
            }
        }
    }
    SS_CUSTOMFUNCTION_STATUS(lpParams) = nStatus;
}
```

**See Also**

[AddCustomFunction](#), [AddCustomFunctionExt](#), [CFGetCellParam](#), [CFGetDoubleParam](#), [CFGetLongParam](#), [CFGetParamInfo](#), [CFGetRangeParam](#), [CFGetStringParam](#), [CFSetResult](#) functions



## GetBottomRightCell Function

[See Also](#)    [Example](#)

### Description

Returns the lower-right visible cell of the spreadsheet.

### Syntax

C++ (OCX)	<code>void CSpreadSheet::GetBottomRightCell (<b>long*</b> Col, <b>long*</b> Row);</code>
C++ (VBX)	<code>void CSpreadSheet::GetBottomRightCell (<b>long FAR*</b> Col, —<b>long FAR*</b> Row);</code>
Visual Basic (OCX)	Function <i>Spread</i> .GetBottomRightCell (Col As <b>Long</b> , Row As <b>Long</b> )
Visual Basic (VBX)	Sub SpreadGetBottomRightCell (SS As <b>Control</b> , Col As <b>Long</b> , —Row As <b>Long</b> )

### Parameters

The following parameters are available:

Parameter	Description
SS	Spread control name
Col	Column number of lower-right cell
Row	Row number of lower-right cell

### Return Type

None

[Print](#)

[Copy](#)

[Close](#)

The following example retrieves the lower-right visible spreadsheet cell.

C++

```
vaSpread1->GetBottomRightCell(&lRightCol, &lBottomRow);
```

OCX

```
vaSpread1.GetBottomRightCell(lRightCol, lBottomRow)
```

Visual Basic

```
SpreadGetBottomRightCell(SS, lRightCol, lBottomRow)
```

**See Also**

[LeftCol](#), [TopRow](#) properties

## GetCellDirtyFlag Function

[See Also](#)    [Example](#)

### Description

Determines if a cell has been modified for a bound Spread control.

### Syntax

C++	<b>BOOL</b> CSpreadSheet::GetCellDirtyFlag ( <b>long</b> Col, <b>long</b> Row);
Visual Basic (OCX)	Function <i>Spread</i> .GetCellDirtyFlag (ByVal Col As <b>Long</b> , —ByVal Row As <b>Long</b> ) As <b>Boolean</b>
Visual Basic (VBX)	Function SpreadGetCellDirtyFlag (SS As <b>Control</b> , —ByVal Col As <b>Long</b> , ByVal Row As <b>Long</b> )

—As **Integer**

### Parameters

The following parameters are available:

Parameter	Description
SS	Spread control name
Col	Column number of cell being checked
Row	Row number of cell being checked

### Remarks

Modified cells are sometimes referred to as "dirty."

### Return Type

VBX: Nonzero if successful; otherwise, 0.  
OCX: TRUE if successful; otherwise, FALSE.

[Print](#)

[Copy](#)

[Close](#)

The following example determines whether the specified cell has been modified.

**C++**

```
BOOL bDirty;  
bDirty = vaSpread1->GetCellDirtyFlag(lCol, lRow);
```

**OCX**

```
vaSpread1.ChangeMade = vaSpread1.GetCellDirtyFlag(Col, Row)
```

**Visual Basic**

```
vaSpread1.ChangeMade = SpreadGetCellDirtyFlag(SS, Col, Row)
```

**See Also**

[SetCellDirtyFlag](#) function

## GetCellFromScreenCoord Function

[See Also](#)    [Example](#)

### Description

Returns the cell coordinates of the cell residing at the specified screen position using screen coordinates.

### Syntax

C++ (OCX)	<code>void CSpreadSheet::GetCellFromScreenCoord (<b>long*</b> Col, <b>long*</b> Row, —<b>long x, long y</b>);</code>
C++ (VBX)	<code>void CSpreadSheet::GetCellFromScreenCoord (<b>long FAR*</b> Col, —<b>long FAR*</b> Row, <b>long x, long y</b>);</code>
Visual Basic (OCX)	Function <i>Spread</i> .GetCellFromScreenCoord (Col As <b>Long</b> , —Row As <b>Long</b> , ByVal x As <b>Long</b> , ByVal y As <b>Long</b> )
Visual Basic (VBX)	Sub SpreadGetCellFromScreenCoord (SS As <b>Control</b> , Col As <b>Long</b> , —Row As <b>Long</b> , ByVal x As <b>Long</b> , ByVal y As <b>Long</b> )

### Parameters

The following parameters are available:

Parameter	Description
SS	Spread control name
Col	Column number of cell
Row	Row number of cell
x	Screen x-coordinate in twips
y	Screen y-coordinate in twips

### Remarks

The returned coordinates are specified in twips.

If you are using C++, you must convert from pixels to twips before calling the GetCellFromScreenCoord function. You can use the following code to perform the conversion.

```
void CMyDlg::OnMouseUpSpread1(short Button, short Shift, long x, long y)
{
    long lCol, lRow;
    long xTwips, yTwips;
    :
    :
    CDC* pdc = this->GetDC( );
    xTwips = x * 1440 / pdc->GetDeviceCaps(LOGPIXELSX);
    yTwips = y * 1440 / pdc->GetDevideCaps(LOGPIXELSY);
    this->ReleaseDC(pdc);
    m_Spread.GetCellFromScreenCoord(&lCol, &lRow, xTwips, yTwips);
    :
}
```

### Return Type

None

[Print](#)

[Copy](#)

[Close](#)

The following example retrieves the cell coordinates of the specified cell using screen coordinates.

C++

```
vaSpread1->GetCellFromScreenCoord(&lCol, &lRow, x, y);
```

OCX

```
vaSpread1.GetCellFromScreenCoord(lCol, lRow, x, y)
```

Visual Basic

```
SpreadGetCellFromScreenCoord(SS, lCol, lRow, x, y)
```



**See Also**

[GetCellPos](#) function

## GetCellPos Function

[See Also](#)      [Example](#)

### Description

Returns the location, and width and height in twips, of the specified cell.

### Syntax

C++ (OCX)                    **BOOL** CSpreadSheet::GetCellPos (**long** Col, **long** Row, **long\*** x,  
                                  —**long\*** y, **long\*** Width, **long\*** Height);

C++ (VBX)                    **BOOL** CSpreadSheet::GetCellPos (**long** Col, **long** Row,  
                                  —**long FAR\*** x, **long FAR\*** y, **long FAR\*** Width,

—**long FAR\*** Height);

Visual Basic (OCX)        Function Spread.GetCellPos (ByVal Col As **Long**,  
                                  —ByVal Row As **Long**, x As **Long**,

—y As **Long**, Width As **Long**, Height As **Long**)

—As **Boolean**

Visual Basic (VBX)        Function SpreadGetCellPos (SS As **Control**, ByVal Col As **Long**,  
                                  —ByVal Row As **Long**, x As **Long**, y As **Long**,

—Width As **Long**, Height As **Long**) As **Integer**

### Parameters

The following parameters are available:

Parameter	Description
SS	Spread control name
Col	Column number of cell
Row	Row number of cell
x	x-coordinate of upper-left corner of cell
y	y-coordinate of upper-left corner of cell
Width	Width of cell
Height	Height of cell

### Remarks

The returned measurements are specified relative to the upper-left corner of the spreadsheet. The returned height and width values are in twips.

### Return Type

VBX: Nonzero if the cell is displayed on the screen; otherwise, 0 and the function does not return coordinates.  
OCX: TRUE if the cell is displayed on the screen; otherwise, FALSE and the function does not return coordinates.

[Print](#)

[Copy](#)

[Close](#)

The following example retrieves the location, width, and height of the specified cell.

C++

```
ret = vaSpread1->GetCellPos(Cellx, Celly, &x, &y, &Width, &Height);
```

OCX

```
ret = vaSpread1.GetCellPos(Cellx, Celly, x, y, Width, Height)
```

Visual Basic

```
ret = SpreadGetCellPos(SS, Cellx, Celly, x, y, Width, Height)
```

**See Also**

[GetCellFromScreenCoord](#) function

## GetClientArea Function

### [Example](#)

#### Description

Returns the width and height of the client area in twips.

#### Syntax

C++ (OCX)	<code>void CSpreadSheet::GetClientArea (<b>long*</b> Width, <b>long*</b> Height);</code>
C++ (VBX)	<code>void CSpreadSheet::GetClientArea (<b>long FAR*</b> Width, —<b>long FAR*</b> Height);</code>
Visual Basic (OCX)	Function <i>Spread</i> .GetClientArea (Width As <b>Long</b> , Height As <b>Long</b> )
Visual Basic (VBX)	Sub SpreadGetClientArea (SS As <b>Control</b> , Width As <b>Long</b> , —Height As <b>Long</b> )

#### Parameters

The following parameters are available:

Parameter	Description
SS	Spread control name
Width	Width of client area
Height	Height of client area

#### Remarks

The client area is the area of the spreadsheet minus the area of the scroll bars.

The returned height and width values are in twips.

#### Return Type

None

[Print](#)

[Copy](#)

[Close](#)

The following example retrieves the width and height of the client area of the Spread control.

C++

```
vaSpread1->GetClientArea(&Width, &Height);
```

OCX

```
vaSpread1.GetClientArea(Width, Height)
```

Visual Basic

```
SpreadGetClientArea(SS, Width, Height)
```

## GetCollItemData Function

[See Also](#)    [Example](#)

### Description

Returns the item data associated with the specified column.

### Syntax

C++	<b>long</b> CSpreadSheet::GetCollItemData ( <b>long</b> Col);
Visual Basic (OCX)	Function <i>Spread</i> .GetCollItemData (ByVal Col As <b>Long</b> ) As <b>Long</b>
Visual Basic (VBX)	Function SpreadGetCollItemData (SS As <b>Control</b> , —ByVal Col As <b>Long</b> ) As <b>Long</b>

### Parameters

The following parameters are available:

Parameter	Description
SS	Spread control name
Col	Column number of column that contains the item data

### Remarks

The item data is a value assigned by the application using the [SetCollItemData](#) function. This value is only for the application's use; the spreadsheet does not use this value.

This function allows an application to associate a specific number with each column. This function is similar to the Visual Basic ItemData property.

### Return Type

Long

[Print](#)

[Copy](#)

[Close](#)

The following example retrieves the item data associated with the fifth spreadsheet column.

C++

```
x = vaSpread1->GetColItemData(5);
```

OCX

```
x = vaSpread1.GetColItemData(5)
```

Visual Basic

```
x = SpreadGetColItemData(SS, 5)
```



**See Also**

[GetItemData](#), [GetRowItemData](#), [SetColItemData](#), [SetItemData](#), [SetRowItemData](#) functions



[Print](#)

[Copy](#)

[Close](#)

The following example retrieves a custom name.

C++

```
CString s;  
s = vaSpread1->GetCustomName("Alpha");
```

OCX

```
Dim s As String  
s = vaSpread1.GetCustomName("Beta")
```

Visual Basic

```
Dim s As String  
s = SpreadGetCustomName(SS, "Gamma")
```

**See Also**

[QueryCustomName](#), [SetCustomName](#) functions



[Print](#)

[Copy](#)

[Close](#)

The following example prefixes all data read into column 2 with an asterisk (\*). When the data is written back to the database, all the asterisks are replaced by tilde characters (~).

#### OCX

```
Sub SS_DataFill (Col As Long, Row As Long, DataType As Integer, fGetData As Integer,
Cancel As Integer)
Dim v As Variant
Dim buf As String
Dim ret As Integer
If fGetData Then
If Col = 2 Then
vaSpread1.Col = Col
vaSpread1.Row = Row
ret = vaSpread1.GetDataFillData(v, 8)
v = "*" + v
vaSpread1.Text = v
Cancel = True
End If
Else If Col = 2 Then
vaSpread1.Col = Col
vaSpread1.Row = Row
v = vaSpread1.Text
Do
ret = InStr(v, "*")
If ret Then
Mid(v, 1, 1) = "~"
Else
Exit Do
End If
Loop
ret = vaSpread1.SetDataFillData(v)
Cancel = True
End If
End If
End Sub
```

#### Visual Basic

```
Sub SS_DataFill (Col As Long, Row As Long, DataType As Integer, fGetData As Integer,
Cancel As Integer)
Dim v As Variant
Dim buf As String
Dim ret As Integer
If fGetData Then
If Col = 2 Then
vaSpread1.Col = Col
vaSpread1.Row = Row
ret = SpreadGetDataFillData(SS, v, 8)
v = "*" + v
vaSpread1.Text = v
Cancel = True
End If
Else If Col = 2 Then
vaSpread1.Col = Col
vaSpread1.Row = Row
v = vaSpread1.Text
Do
ret = InStr(v, "*")
If ret Then
Mid(v, 1, 1) = "~"
Else
Exit Do
End If
End If
```

```
Loop
ret = SpreadSetDataFillData(SS, v)
Cancel = True
End If
End If
End Sub
```

**See Also**

[DataFillEvent](#) property

[DataFill](#) event

[SetDataFillData](#) function



## GetFirstValidCell Function

[See Also](#)    [Example](#)

### Description

Returns the first cell in the spreadsheet that allows the user to move the focus to it.

### Syntax

C++ (OCX)	<code>void CSpreadSheet::GetFirstValidCell (<b>long*</b> Col, <b>long*</b> Row);</code>
C++ (VBX)	<code>void CSpreadSheet::GetFirstValidCell (<b>long FAR*</b> Col, —<b>long FAR*</b> Row);</code>
Visual Basic (OCX)	Function <i>Spread</i> .GetFirstValidCell (Col As <b>Long</b> , Row As <b>Long</b> )
Visual Basic (VBX)	Sub SpreadGetFirstValidCell (SS As <b>Control</b> , Col As <b>Long</b> , —Row As <b>Long</b> )

### Parameters

The following parameters are available:

Parameter	Description
SS	Spread control name
Col	Column number of cell
Row	Row number of cell

### Remarks

If the [EditModePermanent](#) property is set to True, locked and static text cells are not valid active cells; therefore, the first valid cell is not always cell A1. A cell is also not valid if its height or width is set to zero.

### Return Type

None

[Print](#)

[Copy](#)

[Close](#)

The following example retrieves the first spreadsheet cell to which the user can move the focus.

C++

```
vaSpread1->GetFirstValidCell(&Col, &Row);
```

OCX

```
vaSpread1.GetFirstValidCell(Col, Row)
```

Visual Basic

```
SpreadGetFirstValidCell(SS, Col, Row)
```

**See Also**

[CellType](#), [ColWidth](#), [EditModePermanent](#), [Lock](#), [RowHeight](#) properties

[GetLastValidCell](#) function

## GetFormulaSync Function

[See Also](#)

[Example](#)

### Description

Returns whether formulas in the spreadsheet that use relative cell references are adjusted when columns or rows are added or deleted or when blocks of cells are moved or swapped. This function is for VBX controls only.

### Syntax

C++ (VBX)                    **BOOL** CSpreadSheet::GetFormulaSync (*void*);

Visual Basic (VBX)        Function SpreadGetFormulaSync (SS As **Control**) As **Integer**

### Parameters

The following parameter is available:

Parameter	Description
SS	Spread control name

### Remarks

OCX control users should use the [FormulaSync](#) property.

### Return Type

Integer

[Print](#)

[Copy](#)

[Close](#)

The following example determines whether the spreadsheet updates formulas that use relative cell references when rows and columns are inserted or deleted.

C++ (VBX)

```
nSync = vaSpread1->GetFormulaSync( );
```

Visual Basic (VBX)

```
nSync = SpreadGetFormulaSync(SS)
```

**See Also**

[FormulaSync](#) property

[SetFormulaSync](#) function

## GetItemData Function

[See Also](#)    [Example](#)

### Description

Returns the item data associated with a Spread control.

### Syntax

C++                            **long** CSpreadSheet::GetItemData (**void**);  
Visual Basic (OCX)        Function *Spread*.GetItemData ( ) As **Long**  
Visual Basic (VBX)        Function SpreadGetItemData (SS As **Control**) As **Long**

### Parameters

The following parameter is available:

Parameter	Description
SS	Spread control name

### Remarks

The item data is a value assigned by the application using the [SetItemData](#) function. This value is only for the application's use; the spreadsheet does not use this value.

This function allows an application to associate a specific number with each spreadsheet. This function is similar to the Visual Basic ItemData property.

### Return Type

Long

[Print](#)

[Copy](#)

[Close](#)

The following example retrieves the item data associated with a Spread control.

C++

```
x = vaSpread1->GetItemData ( );
```

OCX

```
x = vaSpread1.GetItemData ( )
```

Visual Basic

```
x = SpreadGetItemData (SS)
```



**See Also**

[GetColItemData](#), [GetRowItemData](#), [SetColItemData](#), [SetItemData](#), [SetRowItemData](#) functions

## GetIteration Function

[See Also](#)      [Example](#)

### Description

Returns the number of iterations and the change between iterations for circular references.

### Syntax

C++ (OCX)	<b>BOOL</b> CSpreadSheet::GetIteration ( <b>short*</b> MaxIterations, — <b>double*</b> MaxChange);
C++ (VBX)	<b>short</b> CSpreadSheet::GetIteration ( <b>short FAR*</b> MaxIterations, — <b>double FAR*</b> MaxChange);
Visual Basic (OCX)	Function Spread.GetIteration (MaxIterations As <b>Integer</b> , — MaxChange As <b>Double</b> ) As <b>Boolean</b>
Visual Basic (VBX)	Sub SpreadGetIteration (SS As <b>Control</b> , MaxIterations As <b>Integer</b> , — MaxChange As <b>Double</b> ) As <b>Integer</b>

### Parameters

The following parameters are available:

Parameter	Description
SS	Spread control name
MaxIterations	Maximum number of iterations
MaxChange	Maximum change between iterations

### Remarks

The GetIteration function returns iteration options for circular references in formulas. A circular reference is when a formula in a cell refers to its own cell. For example, the formula (A1\*B1) is circular if it is in cell A1.

The iteration process stops after *MaxIterations* iterations or after all values change by less than the value specified by the *MaxChange* parameter.

### Return Type

VBX: Nonzero if recursive formulas are being evaluated; otherwise, 0.  
OCX: TRUE if recursive formulas are being evaluated; otherwise, FALSE.

[Print](#)

[Copy](#)

[Close](#)

The following example retrieves the maximum number of circular references and maximum change between iterations.

C++

```
vaSpread1->GetIteration(MaxIteration, MaxChange);
```

OCX

```
vaSpread1.GetIteration(MaxIteration, MaxChange)
```

Visual Basic

```
SpreadGetIteration(SS, MaxIteration, MaxChange)
```

**See Also**

[SetIteration](#) function

## GetLastValidCell Function

[See Also](#)    [Example](#)

### Description

Returns the last cell in the spreadsheet to which the user can move the focus.

### Syntax

C++ (OCX)	<code>void CSpreadSheet::GetLastValidCell (<b>long*</b> Col, <b>long*</b> Row);</code>
C++ (VBX)	<code>void CSpreadSheet::GetLastValidCell (<b>long FAR*</b> Col, —<b>long FAR*</b> Row);</code>
Visual Basic (OCX)	Function <i>Spread</i> .GetLastValidCell (Col As <b>Long</b> , Row As <b>Long</b> )
Visual Basic (VBX)	Sub SpreadGetLastValidCell (SS As <b>Control</b> , Col As <b>Long</b> , —Row As <b>Long</b> )

### Parameters

The following parameters are available:

Parameter	Description
SS	Spread control name
Col	Column number of cell
Row	Row number of cell

### Remarks

If the [EditModePermanent](#) property is set to True, locked and static text cells are not valid active cells; therefore, the last valid cell is not always the last cell in the spreadsheet. A cell is also not valid if its height or width is set to zero.

### Return Type

None

[Print](#)

[Copy](#)

[Close](#)

The following example retrieves the last spreadsheet cell to which the user can move the focus.

C++

```
vaSpread1->GetLastValidCell(&Col, &Row);
```

OCX

```
vaSpread1.GetLastValidCell(Col, Row)
```

Visual Basic

```
SpreadGetLastValidCell(SS, Col, Row)
```

**See Also**

[GetFirstValidCell](#) function

## GetMultiSelItem Function

[See Also](#)      [Example](#)

### Description

Determines which rows are selected when the spreadsheet is in multiple-selection or extended-selection operation mode.

### Syntax

C++	<i>long</i> CSpreadSheet::GetMultiSelItem ( <i>long SelPrev</i> );
Visual Basic (OCX)	Function <i>Spread</i> .GetMultiSelItem (ByVal <i>SelPrev</i> As <b>Long</b> ) As <b>Long</b>
Visual Basic (VBX)	Function SpreadGetMultiSelItem (SS As <b>Control</b> , —ByVal <i>SelPrev</i> As <b>Long</b> ) As <b>Long</b>

### Parameters

The following parameters are available:

Parameter	Description
<i>SS</i>	Spread control name
<i>SelPrev</i>	Previously selected row (The first time this function is called, this value must be 0.)

### Remarks

Use the [SelModeSelCount](#) property to determine how many rows have been selected. You must call the function multiple times to retrieve the selected rows.

The *SelPrev* parameter must be set to 0 the first time, then the return value is used in all subsequent calls. When the return value is  $-1$ , there are no more selected rows.

### Return Type

Long



[Print](#)

[Copy](#)

[Close](#)

The following example determines which spreadsheet rows are selected.

C++

```
short index;
long ret, array[100];
ret = 0;
index = 0;
while(1)
{
    ret = vaSpread1->GetMultiSelItem(ret);
    if(ret==-1)
        break;
    array[index++] = ret
}
```

OCX

```
dim index as integer
dim ret as long
dim array(100) as long

ret = 0
index = 0
Do
ret = vaSpread1.GetMultiSelItem(ret)
If ret = -1 then
Exit Do
End If
array(index) = ret
index = index + 1
Loop
```

Visual Basic

```
dim index as integer
dim ret as long
dim array(100) as long

ret = 0
index = 0
Do
ret = SpreadGetMultiSelItem(SS,ret)
If ret = -1 then
Exit Do
End If
array(index) = ret
index = index + 1
Loop
```

**See Also**

[OperationMode](#), [SelModeSelCount](#), [SelModeSelected](#) properties

## GetRefStyle Function

[See Also](#)    [Example](#)

### Description

Returns the reference style used by the spreadsheet to represent formulas and custom names.

### Syntax

C++                                **short** CSpreadSheet::GetRefStyle (**void**);  
Visual Basic (OCX)                Function *Spread*.GetRefStyle ( ) As **Integer**  
Visual Basic (VBX)                Function SpreadGetRefStyle (SS As **Control**) As **Integer**

### Parameters

The following parameter is available:

Parameter	Description
SS	Spread control name

### Remarks

The Spread control can use relative or absolute references for formulas. The default is absolute references that do not change when copied to another position. The default reference style lets the user indicate the current row or column using the number sign (#).

The following values can be returned:

Return Value	Constant	Description	Example
0	SS_REFSTYLE_DEFAULT	(Default) Uses letters and numbers for column and row coordinates; letters and numbers represent absolute coordinates, number sign (#) represents the current column or row	A1, B#, #2
1	SS_REFSTYLE_A1	Uses letters and numbers for column and row coordinates; uses dollar sign (\$) for absolute coordinates	\$A\$1, \$B, \$2, \$B2, C\$5, D4
2	SS_REFSTYLE_R1C1	Uses "R" and number for row, "C" and number for column coordinates; [ ] for relative coordinates, "C" or "R" without a number represent the current row or column	R1C1, RC2, R4C[3], R[2]C4, RC[-2]

SS\_REFSTYLE\_A1 provides relative references. The user can also specify absolute references using a dollar sign before the column or row coordinate. This reference style emulates Microsoft® Excel.

SS\_REFSTYLE\_R1C1 uses row and column number coordinates and allows relative references, which are specified by brackets.

### Return Type

Current reference style.

[Print](#)

[Copy](#)

[Close](#)

The following example retrieves the reference style used by the spreadsheet.

C++

```
Value = vaSpread1->GetRefStyle( );
```

OCX

```
Value = vaSpread1.GetRefStyle( )
```

Visual Basic

```
Value = SpreadGetRefStyle(SS)
```

**See Also**

[SetRefStyle](#) function

## GetRowItemData Function

[See Also](#)    [Example](#)

### Description

Returns the item data associated with the specified row.

### Syntax

C++                            **long** CSpreadSheet::GetRowItemData (**long** Row);  
Visual Basic (OCX)        Function *Spread*.GetRowItemData (ByVal Row As **Long**) As **Long**  
Visual Basic (VBX)        Function SpreadGetRowItemData (SS As **Control**,  
                                  —ByVal Row As **Long**) As **Long**

### Parameters

The following parameters are available:

Parameter	Description
SS	Spread control name
Row	Row number of row that contains the item data

### Remarks

The item data is a value assigned by the application using the [SetRowItemData](#) function. This value is only for the application's use; the spreadsheet does not use this value.

This function allows an application to associate a specific number with each row. This function is similar to the Visual Basic ItemData property.

### Return Type

Long

[Print](#)

[Copy](#)

[Close](#)

The following example retrieves the item data associated with the specified row.

**C++**

```
char buf[20];

for(x = vaSpread1->GetSelBlockRow( );x <= vaSpread1->GetSelBlockRow2( ); x++)
{
    l = vaSpread1->GetRowItemData(x);
    sprintf(buf,"%ld",l);
    vaSpread1->SetRow(x);
    vaSpread1->SetCol(1);
    vaSpread1->SetText(buf);
}
```

**OCX**

```
For x = vaSpread1.SelBlockRow To vaSpread1.SelBlockRow2
    l = vaSpread1.GetRowItemData(x)
    vaSpread1.Text = Str$(l)
    vaSpread1.Row = vaSpread1.Row + 1
Next x
```

**Visual Basic**

```
For x = vaSpread1.SelBlockRow To vaSpread1.SelBlockRow2
    l = SpreadGetRowItemData(SS, x)
    vaSpread1.Text = Str$(l)
    vaSpread1.Row = vaSpread1.Row + 1
Next x
```

**See Also**

[GetColItemData](#), [GetItemData](#), [SetColItemData](#), [SetItemData](#), [SetRowItemData](#) functions



## GetText Function

[See Also](#)    [Example](#)

### Description

Returns the text of the specified cell.

### Syntax

C++ (OCX)                    **BOOL** CSpreadSheet::GetText (**long** Col, **long** Row, **VARIANT\*** Var);

C++ (VBX)                    **CString** CSpreadSheet::GetText (**long** Col, **long** Row);

Visual Basic (OCX)        Function *Spread*.GetText (ByVal Col As **Long**, ByVal Row As **Long**,  
— Var As **Variant**) As **Boolean**

Visual Basic (VBX)        Function SpreadGetText (SS As **Control**, ByVal Col As **Long**,  
—ByVal Row As **Long**, Var As **Variant**) As **Integer**

### Parameters

The following parameters are available:

Parameter	Description
SS	Spread control name
Col	Column number of cell
Row	Row number of cell
Var	Text in cell

### Remarks

Using this function is faster than using the [Text](#) property to retrieve data.

### Return Type

VBX: Nonzero if successful; otherwise, 0.

OCX: TRUE if successful; otherwise, FALSE.

[Print](#)

[Copy](#)

[Close](#)

The following example retrieves the text of the specified cell.

C++

```
char array[10][10][80];
char far *ptr;
CX = vaSpread1->GetSelBlockCol( );
CY = vaSpread1->GetSelBlockRow( );
CXZ = vaSpread1->GetSelBlockCol2( );
CYZ = vaSpread1->GetSelBlockRow2( );
Dify = CYZ - CY;
Difx = CXZ - CX;
for(y = 0; y <= Dify; y++)
    for(x = 0; x <= Difx; x++)
        {
            ptr = vaSpread1->GetText(CX + x, CY + y);
            lstrcpy(array[y][x], ptr);
        }
```

OCX

```
CX = vaSpread1.SelBlockCol
CY = vaSpread1.SelBlockRow
Difx = vaSpread1.SelBlockCol2 - vaSpread1.SelBlockCol
Dify = vaSpread1.SelBlockRow2 - vaSpread1.SelBlockRow
For y = 0 To Dify
For x = 0 To Difx
ret = vaSpread1.GetText(CX + x, CY + y, Array(y, x))
Next x
Next y
```

Visual Basic

```
CX = vaSpread1.SelBlockCol
CY = vaSpread1.SelBlockRow
Difx = vaSpread1.SelBlockCol2 - vaSpread1.SelBlockCol
Dify = vaSpread1.SelBlockRow2 - vaSpread1.SelBlockRow
For y = 0 To Dify
For x = 0 To Difx
ret = SpreadGetText(SS, CX + x, CY + y, Array(y, x))
Next x
Next y
```

**See Also**

[Clip](#), [Text](#) properties

[SetText](#) function

## IsCellSelected Function

### [Example](#)

#### Description

Determines if a cell is selected.

#### Syntax

C++	<b>BOOL</b> CSpreadSheet::IsCellSelected ( <i>long Col, long Row</i> );
Visual Basic (OCX)	Function <i>Spread</i> .IsCellSelected (ByVal <i>Col</i> As <b>Long</b> , —ByVal <i>Row</i> As <b>Long</b> ) As <b>Boolean</b>
Visual Basic (VBX)	Function <i>Spread</i> .IsCellSelected ( <i>SS</i> As <b>Control</b> , ByVal <i>Col</i> As <b>Long</b> , —ByVal <i>Row</i> As <b>Long</b> ) As <b>Integer</b>

#### Parameters

The following parameters are available:

Parameter	Description
<i>SS</i>	Spread control name
<i>Col</i>	Column number of cell
<i>Row</i>	Row number of cell

#### Return Type

VBX: Nonzero if the cell is selected; otherwise, 0.  
OCX: TRUE if the cell is selected; otherwise, FALSE.

[Print](#)

[Copy](#)

[Close](#)

The following example determines if the specified cell is selected.

C++

```
ret = vaSpread1->IsCellSelected(Col, Row);
```

OCX

```
ret = vaSpread1.IsCellSelected(Col, Row)
```

Visual Basic

```
ret = SpreadIsCellSelected(SS, Col, Row)
```

## IsFormulaValid Function

[See Also](#)    [Example](#)

### Description

Determines if a formula string is valid.

### Syntax

C++ (OCX)	<b>BOOL</b> CSpreadSheet::IsFormulaValid ( <b>LPCTSTR</b> <i>Formula</i> );
C++ (VBX)	<b>BOOL</b> CSpreadSheet::IsFormulaValid ( <b>LPCSTR</b> <i>Formula</i> );
Visual Basic (OCX)	Function <i>Spread</i> .IsFormulaValid ( <i>Formula</i> As <b>String</b> ) As <b>Boolean</b>
Visual Basic (VBX)	Function SpreadIsFormulaValid (SS As <b>Control</b> , — <i>Formula</i> As <b>String</b> ) As <b>Integer</b>

### Parameters

The following parameters are available:

Parameter	Description
SS	Spread control name
<i>Formula</i>	Formula string

### Return Type

VBX: Nonzero if the cell formula is valid; otherwise, 0.  
OCX: TRUE if the cell formula is valid; otherwise, FALSE.

[Print](#)

[Copy](#)

[Close](#)

The following example determines whether the specified formula is valid.

C++

```
// Check if the formula is valid
if(vaSpread1->IsFormulaValid(s)==FALSE)
    MessageBox(NULL, s, "Invalid Formula", MB_OK);
```

OCX

```
' Check if the formula is valid
If vaSpread1.IsFormulaValid(s) = False Then
    MsgBox s, 48, "Invalid Formula"
End If
```

Visual Basic

```
' Check if the formula is valid
If SpreadIsFormulaValid(SS, s) = False Then
    MsgBox s, 48, "Invalid Formula"
End If
```

**See Also**

[Formula](#) property



## IsVisible Function

### [Example](#)

#### Description

Determines if the specified cell, row, or column is visible.

#### Syntax

C++	<b>BOOL</b> CSpreadSheet::IsVisible ( <i>long</i> Col, <i>long</i> Row, <b>BOOL</b> Partial);
Visual Basic (OCX)	Function <i>Spread</i> .IsVisible (ByVal Col As <b>Long</b> , ByVal Row As <b>Long</b> , —ByVal Partial As <b>Integer</b> ) As <b>Boolean</b>
Visual Basic (VBX)	Function SpreadIsVisible (SS As <b>Control</b> , ByVal Col As <b>Long</b> , —ByVal Row As <b>Long</b> , ByVal Partial As <b>Integer</b> )

—As **Integer**

#### Parameters

The following parameters are available:

Parameter	Description
SS	Spread control name
Col	Column number of cell
Row	Row number of cell
Partial	Set to True to consider partially displayed cells as visible (Otherwise, a cell must be entirely visible to be considered as visible.)

#### Return Type

VBX: Nonzero if the cell is visible; otherwise, 0.  
OCX: TRUE if the cell is visible; otherwise, FALSE.

[Print](#)

[Copy](#)

[Close](#)

The following example determines whether the specified cell is visible. If the cell is partially displayed, it is considered visible.

C++

```
ret = vaSpread1->IsVisible(Col, Row, TRUE);
```

OCX

```
ret = vaSpread1.IsVisible(Col, Row, True)
```

Visual Basic

```
ret = SpreadIsVisible(SS, Col, Row, True)
```



[Print](#)

[Copy](#)

[Close](#)

The following example loads a binary data file.

C++

```
ret = vaSpread1->LoadFromFile("C:\DATA.SS2");
```

OCX

```
ret = vaSpread1.LoadFromFile("C:\DATA.SS2")
```

Visual Basic

```
ret = SpreadLoadFromFile(SS, "C:\DATA.SS2")
```

**See Also**

[Action](#) (8 - Load) property

[LoadTabFile](#), [SaveTabFile](#), [SaveToFile](#) functions

## LoadTabFile Function

[See Also](#)    [Example](#)

### Description

Loads a tab-delimited data file.

### Syntax

C++ (OCX)	<b>BOOL</b> CSpreadSheet::LoadTabFile ( <b>LPCTSTR</b> FileName);
C++ (VBX)	<b>BOOL</b> CSpreadSheet::LoadTabFile ( <b>LPCSTR</b> FileName);
Visual Basic (OCX)	Function <i>Spread</i> .LoadTabFile (ByVal FileName As <b>String</b> ) As <b>Boolean</b>
Visual Basic (VBX)	Function SpreadLoadTabFile (SS As <b>Control</b> , —ByVal FileName As <b>String</b> ) As <b>Integer</b>

### Parameters

The following parameters are available:

Parameter	Description
SS	Spread control name
FileName	Path and filename of spreadsheet file to load

### Remarks

Use this function to load a tab-delimited data file. Use the [LoadFromFile](#) function to load a binary data file.

Tab-delimited data files contain spreadsheet data separated by tabs and carriage returns. Tab-delimited data files can be opened, modified, and saved using any standard text editor.

### Return Type

TRUE if successful; otherwise, FALSE.

[Print](#)

[Copy](#)

[Close](#)

The following example loads a tab-delimited data file.

C++

```
ret = vaSpread1->LoadTabFile("C:\DATA.TB2", FALSE);
```

OCX

```
ret = vaSpread1.LoadTabFile("C:\DATA.TB2", FALSE)
```

Visual Basic

```
ret = SpreadLoadTabFile(SS, "C:\DATA.TB2", FALSE)
```

**See Also**

[Action](#) (8 - Load) property

[LoadFromFile](#), [SaveTabFile](#), [SaveToFile](#) functions



## QueryCustomName Function

[See Also](#)

### Description

Returns the defined custom names.

### Syntax

C++ (OCX)	<b>CString</b> CSpreadSheet::QueryCustomName ( <b>LPCTSTR</b> Name);
C++ (VBX)	<b>CString</b> CSpreadSheet::QueryCustomName ( <b>LPCSTR</b> PrevName);
Visual Basic (OCX)	Function <i>Spread</i> .QueryCustomName (ByVal <i>PrevName</i> As <b>String</b> ) —As <b>String</b>
Visual Basic (VBX)	Function SpreadQueryCustomName (SS As <b>Control</b> , —ByVal <i>PrevName</i> As <b>String</b> ) As <b>String</b>

### Parameters

The following parameters are available:

#### C++ OCX Parameter

Parameter	Description
<i>Name</i>	Previous custom name

#### VBX, Visual Basic (OCX) Parameters

Parameter	Description
SS	Spread control name
<i>PrevName</i>	Previous custom name

### Remarks

You can define custom names to represent information in the spreadsheet. For example, a custom name can represent a cell, a cell range, or a computed value. Use the [SetCustomName](#) function to define a custom name that can then be used in spreadsheet formulas. The name can be up to 255 characters long.

To return the first custom name, provide an empty string as the value of the *PrevName* parameter.

### Return Type

String

**See Also**

[GetCustomName](#), [SetCustomName](#) functions

## RowHeightToTwips Function

[See Also](#)

[Example](#)

### Description

Converts a row-height measurement in spreadsheet coordinates to twips.

### Syntax

C++ (OCX)	<code>void CSpreadSheet::RowHeightToTwips (<b>long</b> Row, <b>float</b> RowHeight, —<b>long*</b> Twips);</code>
C++ (VBX)	<code>void CSpreadSheet::RowHeightToTwips (<b>long</b> Row, <b>float</b> RowHeight, —<b>long FAR*</b> Twips);</code>
Visual Basic (OCX)	Function <i>Spread</i> .RowHeightToTwips (ByVal Row As <b>Long</b> , —ByVal Height As <b>Single</b> , Twips As <b>Long</b> )
Visual Basic (VBX)	Sub SpreadRowHeightToTwips (SS As <b>Control</b> , ByVal Row As <b>Long</b> , —ByVal RowHeight As <b>Single</b> , Twips As <b>Long</b> )

### Parameters

The following parameters are available:

Parameter	Description
SS	Spread control name
Row	Row number
RowHeight	Height of row
Twips	Height of row in twips

### Return Type

None

[Print](#)

[Copy](#)

[Close](#)

The following example converts a row-height measurement in spreadsheet coordinates to twips.

C++

```
vaSpread1->RowHeightToTwips (Row, Height, Twips);
```

OCX

```
vaSpread1.RowHeightToTwips (Row, Height, Twips)
```

Visual Basic

```
SpreadRowHeightToTwips (SS, Row, Height, Twips)
```

**See Also**

[ColWidthToTwips](#) function

## SaveTabFile Function

[See Also](#)    [Example](#)

### Description

Saves the spreadsheet data to a tab-delimited file.

### Syntax

C++ (OCX)	<b>BOOL</b> CSpreadSheet::SaveTabFile ( <i>LPCTSTR</i> FileName);
C++ (VBX)	<b>BOOL</b> CSpreadSheet::SaveTabFile ( <i>LPCSTR</i> FileName);
Visual Basic (OCX)	Function <i>Spread</i> .SaveTabFile (ByVal <i>FileName</i> As <b>String</b> ) As <b>Boolean</b>
Visual Basic (VBX)	Function SpreadSaveTabFile ( <i>SS</i> As <b>Control</b> , —ByVal <i>FileName</i> As <b>String</b> ) As <b>Integer</b>

### Parameters

The following parameters are available:

Parameter	Description
<i>SS</i>	Spread control name
<i>FileName</i>	Path and filename of file to which to save tab-delimited data

### Remarks

Data for each column is separated by tabs (ASCII 9, or '\t' in C++); rows are separated by carriage returns/line feeds (ASCII 13 and 10, or '\r' and '\n' in C++, respectively). The Spread control saves only text data to the tab-delimited file; the format of the text is not saved.

Tab-delimited files can be opened, modified, and saved using any standard text editor.

By default, tab-delimited spreadsheet files created using Spread or the Spread Designer have the extension .TB2. However, you can use any extension.

### Return Type

VBX: Nonzero if successful; otherwise, 0.  
OCX: TRUE if successful; otherwise, FALSE.

[Print](#)

[Copy](#)

[Close](#)

The following example saves the spreadsheet data to a tab-delimited file.

C++

```
ret = vaSpread1->SaveTabFile("C:\DATA.TB2");
```

OCX

```
ret = vaSpread1.SaveTabFile("C:\DATA.TB2")
```

Visual Basic

```
ret = SpreadSaveTabFile(SS, "C:\DATA.TB2")
```

**See Also**

[Action](#) (10 - Save Values) property

[LoadFromFile](#), [LoadTabFile](#), [SaveToFile](#) functions



## SaveToFile Function

[See Also](#)    [Example](#)

### Description

Saves the spreadsheet data only, or the data with formatting, to a binary file.

### Syntax

C++ (OCX)	<b>BOOL</b> CSpreadSheet::SaveToFile ( <i>LPCTSTR</i> FileName, — <b>BOOL</b> DataOnly);
C++ (VBX)	<b>BOOL</b> CSpreadSheet::SaveToFile ( <i>LPCSTR</i> FileName, — <b>BOOL</b> DataOnly);
Visual Basic (OCX)	Function <i>Spread</i> .SaveToFile (ByVal <i>FileName</i> As <b>String</b> , —ByVal <i>DataOnly</i> As <b>BOOL</b> ) As <b>Boolean</b>
Visual Basic (VBX)	Function SpreadSaveToFile ( <i>SS</i> As <b>Control</b> , ByVal <i>FileName</i> As <b>String</b> , —ByVal <i>DataOnly</i> As <b>Integer</b> ) As <b>Integer</b>

### Parameters

The following parameters are available:

Parameter	Description
<i>SS</i>	Spread control name
<i>FileName</i>	Path and filename of file to which to save binary data
<i>DataOnly</i>	True to save data only; False to save data with formatting

### Remarks

Binary files are in a proprietary format that only the Spread control or the Spread Designer can read. By default, binary spreadsheet files created using Spread or the Spread Designer have the extension .SS2. However, you can use any extension.

If the *DataOnly* parameter is set to True, only the spreadsheet data is saved. This creates a much smaller file; however, formatting information (such as cell types) is not saved. To save the spreadsheet data with formatting, set the *DataOnly* parameter to False.

### Return Type

VBX: Nonzero if successful; otherwise, 0.  
OCX: TRUE if successful; otherwise, FALSE.

[Print](#)

[Copy](#)

[Close](#)

The following example saves the spreadsheet data with formatting to a binary file.

C++

```
ret = vaSpread1->SaveToFile("C:\DATA.SS2", FALSE);
```

OCX

```
ret = vaSpread1.SaveToFile("C:\DATA.SS2", FALSE)
```

Visual Basic

```
ret = SpreadSaveToFile(SS, "C:\DATA.SS2", FALSE)
```

**See Also**

[Action](#) (9 - Save All) property

[LoadFromFile](#), [LoadTabFile](#), [SaveTabFile](#) functions

## SetCellDirtyFlag Function

[See Also](#)    [Example](#)

### Description

Flags a cell or row in a bound Spread control whenever the data in the cell has changed.

### Syntax

C++                            **BOOL** CSpreadSheet::SetCellDirtyFlag (*long Col*, *long Row*,  
                                  —**BOOL Dirty**);

Visual Basic (OCX)        Function *Spread*.SetCellDirtyFlag (ByVal *Col* As **Long**,  
                                  —ByVal *Row* As **Long**, ByVal *Dirty* As **BOOL**)

#### —As **Boolean**

Visual Basic (VBX)        Function SpreadSetCellDirtyFlag (SS As **Control**, ByVal *Col* As **Long**,  
                                  —ByVal *Row* As **Long**, ByVal *Dirty* As **Integer**)

#### —As **Integer**

### Parameters

The following parameters are available:

Parameter	Description
-----------	-------------

SS	Spread control name
----	---------------------

Col	Column of cell to flag (Set this value to —1 to flag the entire row.)
-----	--

Row	Row of cell to flag
-----	---------------------

Dirty	True if the cell or row should be flagged; False if the cell or row should not be flagged
-------	---

### Remarks

This function causes new data to be written to the database when an update is issued. If an entire row is flagged, all cells within that row are written to the database.

When the user changes the data in a cell, the spreadsheet automatically sets the flag. However, if the application changes data in a cell through properties, you must call this function if you want the data written to the database.

### Return Type

VBX: Nonzero if successful; otherwise, 0.

OCX: TRUE if successful; otherwise, FALSE.

[Print](#)

[Copy](#)

[Close](#)

The following example determines whether the cell at column 10, row 20 is flagged, and displays an error message if the cell is not flagged.

C++

```
if(vaSpread1->SetCellDirtyFlag(10, 20, TRUE)==FALSE  
    MessageBox(NULL,"Can't set dirty flag","Error",MB_OK);
```

OCX

```
If vaSpread1.SetCellDirtyFlag(10, 20, True) = False Then  
MsgBox "Can't set dirty flag", 48, "Error"  
End If
```

Visual Basic

```
If SpreadSetCellDirtyFlag(SS, 10, 20, True) = False Then  
MsgBox "Can't set dirty flag", 48, "Error"  
End If
```

**See Also**

[GetCellDirtyFlag](#) function

## SetCollItemData Function

[See Also](#)    [Example](#)

### Description

Sets the item data associated with the specified column.

### Syntax

C++	<code>void CSpreadSheet::SetCollItemData (<b>long</b> Col, <b>long</b> Value);</code>
Visual Basic (OCX)	Function <i>Spread</i> .SetCollItemData (ByVal Col As <b>Long</b> , —ByVal Var As <b>Long</b> )
Visual Basic (VBX)	Sub SpreadSetCollItemData (SS As <b>Control</b> , ByVal Col As <b>Long</b> , —ByVal Var As <b>Long</b> )

### Parameters

The following parameters are available:

#### C++ Parameters

Parameter	Description
<i>Col</i>	Column number of column that contains the item data
<i>Value</i>	Item data

#### OCX, VBX Parameters

Parameter	Description
<i>SS</i>	Spread control name
<i>Col</i>	Column number of column that contains the item data
<i>Var</i>	Item data

### Remarks

The item data is a value retrieved by the application using the [GetCollItemData](#) function. This value is only for the application's use; the spreadsheet does not use this value.

This function allows an application to associate a specific number with each column. This function is similar to the Visual Basic ItemData property.

### Return Type

None

[Print](#)

[Copy](#)

[Close](#)

The following example sets the item data associated with the specified column.

**C++**

```
for(x = vaSpread1->GetSelBlockCol( )
  x<=vaSpread1->SelBlockCol2( ); x++)
{
  vaSpread1->SetColItemData(x, 1);
}
```

**OCX**

```
For x = vaSpread1.SelBlockCol To vaSpread1.SelBlockCol2
  l = Val(vaSpread1.Text)
  vaSpread1.SetColItemData(x, l)
  vaSpread1.Row = vaSpread1.Row + 1
Next x
```

**Visual Basic**

```
For x = vaSpread1.SelBlockCol To vaSpread1.SelBlockCol2
  l = Val(vaSpread1.Text)
  SpreadSetColItemData(SS, x, l)
  vaSpread1.Row = vaSpread1.Row + 1
Next x
```



**See Also**

[GetColItemData](#), [GetItemData](#), [GetRowItemData](#), [SetItemData](#), [SetRowItemData](#) functions

## SetCustomName Function

[See Also](#)      [Example](#)

### Description

Sets a custom name and its value.

### Syntax

C++ (OCX)	<b>BOOL</b> CSpreadSheet::SetCustomName ( <b>LPCTSTR</b> Name, — <b>LPCTSTR</b> Value);
C++ (VBX)	<b>BOOL</b> CSpreadSheet::SetCustomName ( <b>LPCSTR</b> Name, — <b>LPCSTR</b> Value);
Visual Basic (OCX)	Function <i>Spread</i> .SetCustomName (ByVal <i>Name</i> As <b>String</b> , —ByVal <i>Value</i> As <b>String</b> ) As <b>Boolean</b>
Visual Basic (VBX)	Function SpreadSetCustomName ( <i>SS</i> As <b>Control</b> , —ByVal <i>Name</i> As <b>String</b> , ByVal <i>Value</i> As <b>String</b> )

—As **Integer**

### Parameters

The following parameters are available:

Parameter	Description
<i>SS</i>	Spread control name
<i>Name</i>	Custom name
<i>Value</i>	Value represented by custom name

### Remarks

You can define custom names to represent information in the spreadsheet. For example, a custom name can represent a cell, a cell range, or a computed value.

Use the SetCustomName function to define a custom name that can then be used in spreadsheet formulas. You can redefine the custom name by assigning a new value to the name, or you can undefine it by assigning the NULL value to it. You cannot undefine a name if it is used in a spreadsheet expression.

The *Name* string can contain up to 255 characters and can include letters, numbers, or underscores. The first character must be a letter or an underscore.

The *Value* parameter can represent a constant, a cell, a range of cells, or a computed value.

### Return Type

VBX: Nonzero if the function completes successfully; otherwise, 0. A zero value is returned if you try to set the value to an invalid expression or if you try to undefine an existing custom name used in an expression.

OCX: TRUE if the function completes successfully; otherwise, FALSE. A FALSE value is returned if you try to set the value to an invalid expression or if you try to undefine an existing custom name used in an expression.

[Print](#)

[Copy](#)

[Close](#)

The following example defines custom names for a value, a cell, a range of cells, and an expression. The custom name "Alpha" is undefined, and the example returns the value assigned the name "Beta."

C++

```
CString BetaValue;
// define constant
vaSpread1->SetCustomName("Alpha", "5");
// define cell
vaSpread1->SetCustomName("Cell", "B1");
// define cell range
vaSpread1->SetCustomName("Table", "A1:B1");
// define expression
vaSpread1->SetCustomName("Total", "sum(Table)");
// undefine name
vaSpread1->SetCustomName("Alpha", NULL);
// return value
BetaValue = GetCustomName("Beta");
```

OCX

```
Dim BetaValue As String
' define constant
vaSpread1.SetCustomName("Alpha", "5")
' define cell
vaSpread1.SetCustomName("Cell", "B1")
' define cell range
vaSpread1.SetCustomName("Table", "A1:B1")
' define expression
vaSpread1.SetCustomName("Total", "sum(Table)")
' undefine name
vaSpread1.SetCustomName("Alpha", NULL)
' return value
BetaValue = vaSpread1.GetCustomName("Beta")
```

Visual Basic

```
Dim BetaValue As String
' define constant
SpreadSetCustomName(SS, "Alpha", "5")
' define cell
SpreadSetCustomName(SS, "Cell", "B1")
' define cell range
SpreadSetCustomName(SS, "Table", "A1:B1")
' define expression
SpreadSetCustomName(SS, "Total", "sum(Table)")
' undefine name
SpreadSetCustomName(SS, "Alpha", NULL)
' return value
BetaValue = SpreadGetCustomName(SS, "Beta")
```

**See Also**

[GetCustomName](#), [QueryCustomName](#) functions

## SetDataFillData Function

[See Also](#)      [Example](#)

### Description

Sets data in the database inside the [DataFill](#) event.

### Syntax

Visual Basic (OCX)      Function *Spread.SetDataFillData* (*Var* As **Variant**) As **Boolean**

Visual Basic (VBX)      Function *SpreadSetDataFillData* (*SS* As **Control**,  
— *Var* As **Variant**) As **Integer**

### Parameters

The following parameters are available:

Parameter	Description
<i>SS</i>	Spread control name
<i>Var</i>	Value written to the database

### Remarks

This function lets the application interact with any data contained in the spreadsheet before it is written to the database.

### Return Type

VBX: Nonzero if successful; otherwise, 0.  
OCX: TRUE if successful; otherwise, FALSE.

[Print](#)

[Copy](#)

[Close](#)

The following example prefixes all data read into column 2 with an asterisk (\*). When the data is written to the database, all the asterisks are replaced with tilde characters (~).

OCX

```
Sub SS_DataFill (Col As Long, Row As Long, DataType As Integer, fGetData As Integer,
Cancel As Integer)
Dim v As Variant
Dim buf As String
Dim ret As Integer
If fGetData Then
    If Col = 2 Then
        vaSpread1.Col = Col
        vaSpread1.Row = Row
        ret = vaSpread1.GetDataFillData(v, 8)
        v = "*" + v
        vaSpread1.Text = v
        Cancel = True
    End If
Else
    If Col = 2 Then
        vaSpread1.Col = Col
        vaSpread1.Row = Row
        v = vaSpread1.Text
        Do
            ret = InStr(v, "*")
            If ret Then
                Mid(v, 1, 1) = "~"
            Else
                Exit Do
            End If
        Loop
        ret = vaSpread1.SetDataFillData(v)
        Cancel = True
    End If
End If
End Sub
```

Visual Basic

```
Sub SS_DataFill (Col As Long, Row As Long, DataType As Integer, fGetData As Integer,
Cancel As Integer)
Dim v As Variant
Dim buf As String
Dim ret As Integer
If fGetData Then
    If Col = 2 Then
        vaSpread1.Col = Col
        vaSpread1.Row = Row
        ret = SpreadGetDataFillData(SS, v, 8)
        v = "*" + v
        vaSpread1.Text = v
        Cancel = True
    End If
Else
    If Col = 2 Then
        vaSpread1.Col = Col
        vaSpread1.Row = Row
        v = vaSpread1.Text
        Do
            ret = InStr(v, "*")
        Loop
        If ret Then
            Mid(v, 1, 1) = "~"
        Else
        End If
    End If
End Sub
```

```
        Exit Do
    End If
Loop
    ret = SpreadSetDataFillData(SS, v)
    Cancel = True
End If
End If
End Sub
```

**See Also**

[DataFillEvent](#) property

[DataFill](#) event

[GetDataFillData](#) function





[Print](#)

[Copy](#)

[Close](#)

The following example sets the spreadsheet to not update formulas when rows and columns are inserted or deleted.

C++ (VBX)

```
nSync = vaSpread1->SetFormulaSync (FALSE) ;
```

Visual Basic (VBX)

```
nSync = SpreadSetFormulaSync (SS, FALSE)
```

**See Also**

[FormulaSync](#) property

[GetFormulaSync](#) function

## SetItemData Function

[See Also](#)    [Example](#)

### Description

Sets the item data associated with a Spread control.

### Syntax

C++	<code>void CSpreadSheet::SetItemData (<b>long</b> Value);</code>
Visual Basic (OCX)	Function <i>Spread</i> .SetItemData (ByVal <i>Var</i> As <b>Long</b> )
Visual Basic (VBX)	Sub SpreadSetItemData (SS As <b>Control</b> , ByVal <i>Var</i> As <b>Long</b> )

### Parameters

The following parameters are available:

#### C++ Parameter

Parameter	Description
<i>Value</i>	Item data

#### OCX, VBX Parameters

Parameter	Description
SS	Spread control name
<i>Var</i>	Item data

### Remarks

The item data is a value retrieved by the application using the [GetItemData](#) function. This value is only for the application's use; the spreadsheet does not use this value.

This function allows an application to associate a specific number with each spreadsheet. This function is similar to the Visual Basic ItemData property.

### Return Type

None

[Print](#)

[Copy](#)

[Close](#)

The following example sets the item data associated with a Spread control.

C++

```
vaSpread1->SetItemData(1);
```

OCX

```
vaSpread1.SetItemData(1)
```

Visual Basic

```
SpreadSetItemData(SS, 1)
```

**See Also**

[GetColItemData](#), [GetItemData](#), [GetRowItemData](#), [SetColItemData](#), [SetRowItemData](#) functions

## SetIteration Function

[See Also](#)      [Example](#)

### Description

Sets the number of iterations and the change between iterations for circular references.

### Syntax

C++                            **void** CSpreadSheet::SetIteration (**BOOL** *Iteration*,  
                                  — **short** *MaxIterations*, **double** *MaxChange*);

Visual Basic (OCX)        Function *Spread*.SetIteration (ByVal *Iteration* As **BOOL**,  
                                  —ByVal *MaxIterations* As **Integer**,

—ByVal *MaxChange* As **Double**)

Visual Basic (VBX)        Sub SpreadSetIteration (SS As **Control**, ByVal *Iteration* As **Integer**,  
                                  —ByVal *MaxIterations* As **Integer**,

—ByVal *MaxChange* As **Double**)

### Parameters

The following parameters are available:

Parameter	Description
<i>SS</i>	Spread control name
<i>Iterations</i>	True if circular references are evaluated; False if circular references are not evaluated
<i>MaxIterations</i>	Maximum number of iterations
<i>MaxChange</i>	Maximum change between iterations

### Remarks

The SetIteration function sets iteration options for circular references in formulas. A circular reference is when a formula in a cell refers to its own cell. For example, the formula (A1\*B1) is circular if it is in cell A1.

The iteration process stops after *MaxIterations* iterations or after all values change by less than the value specified by the *MaxChange* parameter.

### Return Type

Nonzero if successful; otherwise, 0.

[Print](#)

[Copy](#)

[Close](#)

The following example sets the evaluation for circular references to stop after 5 iterations and change less than 0.01.

C++

```
vaSpread1->SetIteration(TRUE, 5, 0.01);
```

OCX

```
vaSpread1.SetIteration(TRUE, 5, 0.01)
```

Visual Basic

```
SpreadSetIteration(SS, TRUE, 5, 0.01)
```



**See Also**

[GetIteration](#) function

# SetRefStyle Function

[See Also](#)      [Example](#)

## Description

Sets the reference style used by the spreadsheet to represent formulas and custom names.

## Syntax

C++                                **void** CSpreadSheet::SetRefStyle (*short* RefStyle);  
Visual Basic (OCX)            Function *Spread*.SetRefStyle (ByVal RefStyle As **Integer**)  
Visual Basic (VBX)            Sub SpreadSetRefStyle (SS As **Control**, ByVal RefStyle As **Integer**)

## Parameters

The following parameters are available:

Parameter	Description
-----------	-------------

SS	Spread control name
----	---------------------

RefStyle	Reference style
----------	-----------------

Use one of the following values:

Value	Description	Example
SS_REFSTYLE_DEFAULT	(Default) Uses letters and numbers for column and row coordinates; letters and numbers represent absolute coordinates, number sign (#) represents the current column or row	A1, B#, #2
SS_REFSTYLE_A1	Uses letters and numbers for column and row coordinates; uses dollar sign (\$) for absolute coordinates	\$A\$1, \$B, \$2, \$B2, C\$5, D4
SS_REFSTYLE_R1C1	Uses "R" and number for row, "C" and number for column coordinates; [ ] for relative coordinates, "C" or "R" without a number represent the current row or column	R1C1, RC2, R4C[3], RC[-2]

## Remarks

The Spread control can use relative or absolute references for formulas. The default is absolute references that do not change when copied to another position. The default reference style lets the user indicate the current row or column using the number sign (#).

SS\_REFSTYLE\_A1 provides relative references. The user can also specify absolute references using a dollar sign before the column or row coordinate. This reference style emulates Microsoft® Excel.

SS\_REFSTYLE\_R1C1 uses row and column number coordinates and allows relative references, which are specified by brackets.

## Return Type

Current reference style.

[Print](#)

[Copy](#)

[Close](#)

The following example sets the reference style used by the spreadsheet.

C++

```
vaSpread1->SetRefStyle(SS_REFSTYLE_DEFAULT);
```

OCX

```
vaSpread1.SetRefStyle(SS_REFSTYLE_DEFAULT)
```

Visual Basic

```
SpreadSetRefStyle(SS, SS_REFSTYLE_DEFAULT)
```

**See Also**

[GetRefStyle](#) function

## SetRowItemData Function

[See Also](#)    [Example](#)

### Description

Sets the item data associated with the specified row.

### Syntax

C++	<code>void CSpreadSheet::SetRowItemData (<i>long</i> Row, <i>long</i> Value);</code>
Visual Basic (OCX)	Function <i>Spread</i> .SetRowItemData (ByVal Row As <b>Long</b> , —ByVal Var As <b>Long</b> )
Visual Basic (VBX)	Sub SpreadSetRowItemData (SS As <b>Control</b> , ByVal Row As <b>Long</b> , —ByVal Var As <b>Long</b> )

### Parameters

The following parameters are available:

#### C++ Parameters

Parameter	Description
<i>Row</i>	Row to contain the item data
<i>Value</i>	Item data

#### OCX, VBX Parameters

Parameter	Description
SS	Spread control name
<i>Row</i>	Row to contain the item data
<i>Var</i>	Item data

### Remarks

The item data is a value retrieved by the application using the [GetRowItemData](#) function. This value is only for the application's use; the spreadsheet does not use this value.

This function allows an application to associate a specific number with each row. This function is similar to the Visual Basic ItemData property.

### Return Type

None

[Print](#)

[Copy](#)

[Close](#)

The following example sets the item data associated with the specified row.

**C++**

```
for(x = vaSpread1->GetSelBlockRow( );  
x <= vaSpread1->GetSelBlockRow2( ); x++)  
{  
vaSpread1->SetRowItemData(x, 1);  
}
```

**OCX**

```
For x = FP_Case.vaSpread1.SelBlockRow To FP_Case.vaSpread1.SelBlockRow2  
l = Val(vaSpread1.Text)  
vaSpread1.SetRowItemData FP_Case.x, l  
vaSpread1.Row = vaSpread1.Row + 1  
Next x
```

**Visual Basic**

```
For x = FP_Case.vaSpread1.SelBlockRow To FP_Case.vaSpread1.SelBlockRow2  
l = Val(vaSpread1.Text)  
SpreadSetRowItemData FP_Case.vaSpread1, x, l  
vaSpread1.Row = vaSpread1.Row + 1  
Next x
```

**See Also**

[GetColItemData](#), [GetItemData](#), [GetRowItemData](#), [SetColItemData](#), [SetItemData](#) functions

## SetText Function

[See Also](#)      [Example](#)

### Description

Sets the contents of the specified cell.

### Syntax

C++ (OCX)	<code>void CSpreadSheet::SetText (<b>long</b> Col, <b>long</b> Row, <b>VARIANT*</b> Var);</code>
C++ (VBX)	<code>void CSpreadSheet::SetText (<b>long</b> Col, <b>long</b> Row, <b>LPCSTR</b> lpszText);</code>
Visual Basic (OCX)	Function <i>Spread</i> .SetText (ByVal <i>Col</i> As <b>Long</b> , ByVal <i>Row</i> As <b>Long</b> , — <i>Var</i> As <b>Variant</b> )
Visual Basic (VBX)	Sub SpreadSetText (SS As <b>Control</b> , ByVal <i>Col</i> As <b>Long</b> , —ByVal <i>Row</i> As <b>Long</b> , <i>Var</i> As <b>Variant</b> )

### Parameters

The following parameters are available:

#### C++ (VBX) Parameters

Parameter	Description
<i>Col</i>	Column number of cell to contain data
<i>Row</i>	Row number of cell to contain data
<i>lpszText</i>	Text being assigned to cell

#### OCX, Visual Basic (VBX) Parameters

Parameter	Description
SS	Spread control name
<i>Col</i>	Column number of cell to contain data
<i>Row</i>	Row number of cell to contain data
<i>Var</i>	Text being assigned to cell

### Remarks

Using this function is faster than using the [Text](#) property to assign data.

If you are supplying data for a time cell and the time value includes an a.m./p.m. indicator, include a space between the time value and the indicator, for example, "10:00:00 p.m."

### Return Type

None



[Print](#)

[Copy](#)

[Close](#)

The following example sets the contents of the specified cell.

C++

```
vaSpread1->SetReDraw(FALSE);
for(y = 0; y <= Dify; y++)
for(x = 0; x <= Difx; x++)
    vaSpread1->SetText(CX + x, CY + y, Array[y][x]);
vaSpread1->SetReDraw(TRUE);
```

OCX

```
CX = vaSpread1.SelBlockCol
CY = vaSpread1.SelBlockRow
Difx = vaSpread1.SelBlockCol2 - vaSpread1.SelBlockCol
Dify = vaSpread1.SelBlockRow2 - vaSpread1.SelBlockRow
vaSpread1.ReDraw = False
For y = 0 To Dify
For x = 0 To Difx
    vaSpread1.SetText(CX + x, CY + y, Array(y, x))
Next x
Next y
vaSpread1.ReDraw = True
```

Visual Basic

```
CX = vaSpread1.SelBlockCol
CY = vaSpread1.SelBlockRow
Difx = vaSpread1.SelBlockCol2 - vaSpread1.SelBlockCol
Dify = vaSpread1.SelBlockRow2 - vaSpread1.SelBlockRow
vaSpread1.ReDraw = False
For y = 0 To Dify
For x = 0 To Difx
    SpreadSetText(SS, CX + x, CY + y, Array(y, x))
Next x
Next y
vaSpread1.ReDraw = True
```

**See Also**

[GetText](#) function

## TwipsToColWidth Function

[See Also](#)    [Example](#)

### Description

Converts a twips measurement to a spreadsheet column-width measurement.

### Syntax

C++ (OCX)	<code>void CSpreadSheet::TwipsToColWidth (<b>long</b> Twips, <b>float*</b> ColWidth);</code>
C++ (VBX)	<code>void CSpreadSheet::TwipsToColWidth (<b>long</b> Twips, — <b>float FAR*</b> ColWidth);</code>
Visual Basic (OCX)	Function <i>Spread</i> .TwipsToColWidth (ByVal Twips As <b>Long</b> , — ColWidth As <b>Single</b> )
Visual Basic (VBX)	Sub SpreadTwipsToColWidth (SS As <b>Control</b> , ByVal Twips As <b>Long</b> , — ColWidth As <b>Single</b> )

### Parameters

The following parameters are available:

Parameter	Description
SS	Spread control name
Twips	Twips measurement
ColWidth	Column-width measurement

### Return Type

None

[Print](#)

[Copy](#)

[Close](#)

The following example converts a twips measurement to a spreadsheet column-width measurement.

C++

```
vaSpread1->TwipsToColWidth(Twips, vaSpread1->GetColWidth( ));
```

OCX

```
vaSpread1.TwipsToColWidth(Twips, vaSpread1.ColWidth)
```

Visual Basic

```
SpreadTwipsToColWidth(SS, Twips, vaSpread1.ColWidth)
```

**See Also**

[ColWidthToTwips](#) function

## TwipsToRowHeight Function

[See Also](#)    [Example](#)

### Description

Converts a twips measurement to a spreadsheet row-height measurement.

### Syntax

C++ (OCX)	<code>void CSpreadSheet::TwipsToRowHeight (<b>long</b> Row, <b>long</b> Twips, —<b>float*</b> RowHeight);</code>
C++ (VBX)	<code>void CSpreadSheet::TwipsToRowHeight (<b>long</b> Row, <b>long</b> Twips, —<b>float FAR*</b> RowHeight);</code>
Visual Basic (OCX)	Function <i>Spread</i> .TwipsToRowHeight (ByVal Row As <b>Long</b> , —ByVal Twips As <b>Long</b> , RowHeight As <b>Single</b> )
Visual Basic (VBX)	Sub SpreadTwipsToRowHeight (SS As <b>Control</b> , —ByVal Row As <b>Long</b> , ByVal Twips As <b>Long</b> ,

—RowHeight As **Single**)

### Parameters

The following parameters are available:

Parameter	Description
<i>SS</i>	Spread control name
<i>Row</i>	Row number
<i>Twips</i>	Twips measurement
<i>RowHeight</i>	Row height measurement

### Return Type

None

[Print](#)

[Copy](#)

[Close](#)

The following example converts a twips measurement to a spreadsheet row-height measurement.

C++

```
vaSpread1->TwipsToRowHeight(vaSpread1->GetActiveRow( ), Twips,  
vaSpread1.GetRowHeight( ));
```

OCX

```
vaSpread1.TwipsToRowHeight(vaSpread1.ActiveRow, Twips, vaSpread1.RowHeight)
```

Visual Basic

```
SpreadTwipsToRowHeight(SS, vaSpread1.ActiveRow, Twips, vaSpread1.RowHeight)
```

**See Also**

[RowHeightToTwips](#) function



## Introducing Spread

- [Introduction](#)
- [Changes from Previous Versions](#)
- [Upgrading from Previous Versions of Spread/VBX, Spread/VBX++, and Grid/VBX](#)
- [Documentation Conventions](#)
- [Contacting FarPoint Technologies, Inc.](#)

## Introduction

Spread offers a versatile and easy-to-use spreadsheet control for your interface. The Spread package offers DLL, VBX, and OCX control types so you can easily work in almost any development environment.

The Spread control has a unique Spread Designer, which extends the design environment by allowing you complete access to and control over most of the spreadsheet's properties. With a completely redesigned interface, including property page—like interaction, you can experiment with the spreadsheet design and look. You can create and save templates that contain your own spreadsheet designs. You can record actions you perform in the Spread Designer and generate documented C++ and Visual Basic code. Together these features of the Spread Designer provide a powerful design tool for the seasoned or beginning developer.

Use Spread to create custom, powerful spreadsheets quickly and easily. Give your application an exciting look with this easy-to-use product from FarPoint Technologies.

## Changes from Previous Versions

### Data Binding

The Q+E Database Library product is no longer supported.

### Spread Designer

The Interface Designer has been renamed the Spread Designer. However, you access the designer the same way (by double-clicking the InterfaceDesigner property in the Properties window).

### Changes to Properties

- [Action](#) property setting 8 (Load) is available for VBX controls in Visual Basic and Microsoft Visual C++ only. OCX and Borland C++ users must use the [LoadFromFile](#) or [LoadTabFile](#) method or function.
- [Action](#) property setting 9 (Save All) is available for VBX controls in Visual Basic and Microsoft Visual C++ only. OCX and Borland C++ users must use the [SaveToFile](#) method or function.
- [Action](#) property setting 10 (Save Values) is available for VBX controls in Visual Basic and Microsoft Visual C++ only. OCX and Borland C++ users must use the [SaveTabFile](#) method or function.
- [Action](#) property setting 31 (Refresh Bound) is obsolete. The Q+E Database Library product is no longer supported.
- The CalcDependencies property is obsolete but is still supported for backward compatibility. Now, the dependencies tables are refreshed by the Spread control automatically as needed.
- The [CellBorderStyle](#) property has five new settings: 11 (Fine Solid), 12 (Fine Dash), 13 (Fine Dot), 14 (Fine Dash Dot), and 15 (Fine Dash Dot Dot).
- The DataConnect and DataSelect properties are obsolete but are still supported for backward compatibility. The Q+E Database Library product is no longer supported.
- The LoadFile, LoadTabFile, SaveFile, and SaveTabFile properties are not documented but are still supported for backward compatibility.

**Note** If you are planning to migrate to Spread OCX controls, for best results use the [SpreadLoadFromFile](#), [SpreadLoadTabFile](#), [SpreadSaveToFile](#), and [SpreadSaveTabFile](#) functions.

### New Properties

The following properties have been added:

Property	Description
<a href="#">BackColorStyle</a>	Sets or returns whether the background colors of cells overlap the grid lines
<a href="#">CursorIcon</a>	Specifies the icon to display as the mouse pointer
<a href="#">DataChanged</a>	Determines whether data in the Spread control has been changed by some process other than retrieving data from the current record
<a href="#">FormulaSync</a>	Sets or returns whether formulas in the spreadsheet are adjusted when columns or rows are added or deleted or when blocks of cells are moved or swapped (This property is for OCX controls only.)
<a href="#">PrintOrientation</a>	Specifies the page orientation (portrait, landscape, or default printer setting) when printing the spreadsheet

### Changes to Events

The [EditError](#) event has 14 new error codes.

### Changes to Functions

The GetDataConnectHandle and GetDataSelectHandle functions are obsolete. The Q+E Database Library product is no longer supported.

### New Functions

The following functions have been added:

Function	Description
<a href="#">SpreadAddCustomFunctionExt</a>	Adds a custom function with a variable number of parameters
<a href="#">SpreadCFGetCellParam</a>	Returns the column and row coordinates of a cell reference parameter that is being passed into a custom function
<a href="#">SpreadCFGetDoubleParamExt</a>	Returns a custom functions parameter as a double-precision, floating-point value (Borland C++ users must use this function instead of the <a href="#">SpreadCFGetDoubleParam</a> function.)
<a href="#">SpreadCFGetRangeParam</a>	Returns the column and row coordinates of a range reference

<a href="#"><u>SpreadDerefHlstrLen</u></a>	parameter that is being passed into a custom function Returns a pointer to the string data of a Basic language string and the length of the string (This function is for MFC VBX users only.)
<a href="#"><u>SpreadGetCustomName</u></a>	Returns the value of a custom name
<a href="#"><u>SpreadGetFormulaSync</u></a>	Returns whether formulas in the spreadsheet that use absolute cell references are adjusted when columns or rows are added or deleted or when blocks of cells are moved or swapped (This function is for VBX controls only.)
<a href="#"><u>SpreadGetIteration</u></a>	Returns the number of iterations and the change between iterations for circular references
<a href="#"><u>SpreadGetRefStyle</u></a>	Returns the reference style used by the spreadsheet to represent formulas and custom names
<a href="#"><u>SpreadLoadFromFile</u></a>	Loads a previously saved spreadsheet from a binary data file
<a href="#"><u>SpreadLoadTabFile</u></a>	Loads a previously saved spreadsheet from a tab-delimited data file
<a href="#"><u>SpreadQueryCustomName</u></a>	Returns the defined custom names
<a href="#"><u>SpreadSaveToFile</u></a>	Saves the spreadsheet data as a binary file
<a href="#"><u>SpreadSetCustomName</u></a>	Sets a custom name and its value
<a href="#"><u>SpreadSetFormulaSync</u></a>	Adjusts formulas in the spreadsheet that use absolute cell references when columns or rows are added or deleted or when blocks of cells are moved or swapped (This function is for VBX controls only.)
<a href="#"><u>SpreadSetIteration</u></a>	Sets the number of iterations and the change between iterations for circular references
<a href="#"><u>SpreadSetRefStyle</u></a>	Sets the reference style used by the spreadsheet to represent formulas and custom names

## Upgrading from Previous Versions of Spread/VBX, Spread/VBX++, and Grid/VBX

- [Upgrading Existing Spread/VBX Version 2.1 Projects](#)
- [Upgrading Existing Spread/VBX++ Version 2.0 Projects](#)
- [Upgrading Existing Grid/VBX Version 1.0 Projects](#)

## Upgrading Existing Spread/VBX Version 2.1 Projects

If you have created applications using Spread/VBX version 2.1, you must upgrade your existing projects. Complete the steps provided for your development environment.

- [Upgrading Visual Basic Projects to Use the VBX Control](#)
- [Upgrading Visual Basic Projects to Use the OCX Controls](#)

## Upgrading Visual Basic Projects to Use the VBX Control

If you formerly used Spread/VBX version 2.1 and now want to use the version 2.5 VBX control, complete the following steps.

### To upgrade existing Visual Basic projects for the VBX control

1. Open your existing project.
2. If the project used the file VB\_CONST.TXT, use the file SSVBX.BAS in place of it.
3. Save the project file and form files as text.
4. Edit the project file (.MAK) in a text editor to change the following references:  

<b>Change . . .</b>	<b>To . . .</b>
SPREAD20.VBX	SSVBX25.VBX
5. Edit the form files (.FRM) in a text editor to change all Spread class references as follows:  

<b>Change . . .</b>	<b>To . . .</b>
Spreadsheet	vaSpread
6. Load your project and verify your changes.

## Upgrading Visual Basic Projects to Use the OCX Controls

If you formerly used Spread/VBX version 2.1 and now want to use the version 2.5 OCX control (16- or 32-bit), complete the following steps.

### To upgrade existing Visual Basic projects for OCX controls

1. Open your existing project.
2. If the project used the file VB\_CONST.TXT, use the file SSOCX.BAS in place of it.
3. Save the project file and form files as text.
4. Edit the project file (.MAK) in a text editor to change the following references:  

Change . . .	To . . .
SPREAD20.VBX	SSVBX25.VBX
5. Edit the form files (.FRM) in a text editor to change all Spread class references as follows:  

Change . . .	To . . .
Spreadsheet	vaSpread
6. Load your project and verify your changes.
7. Save the project to rewrite the binary data in the .FRX file.
8. Start Visual Basic 4.0 (16-bit) or Visual Basic 4.0 (32-bit) and load the project file (.MAK).
9. For each program file (.FRM) containing a spreadsheet, update the event parameters as needed. See the file EVENTS.TXT for the parameter changes needed for using the events with the OCX control.
10. For each program file (.FRM) containing a spreadsheet, functions must be changed to methods. Compare the following code:

#### Visual Basic code using Spread/VBX 2.0

```
bSuccess = SpreadAddCustomFunction(Spread1, "Cube", 1)
bValid = SpreadIsFormulaValid(Spread1, "a1+b1")
SpreadSetText(Spread1, Col, Row, "Text")
```

#### Visual Basic code using the OCX control

```
bSuccess = Spread1.AddCustomFunction("Cube", 1)
bValid = Spread1.IsFormulaValid("a1+b1")
Spread1.SetText Col, Row, "Text"
```

## Upgrading Existing Spread/VBX++ Version 2.0 Projects

If you have created applications using Spread/VBX++ version 2.0 and now want to use the version 2.5 VBX control, you must upgrade your existing projects. Complete the steps provided for your development environment.

### To upgrade existing C++ projects for the VBX control

1. Open your existing project.
2. If the project used the file SSPP.H, use the file SSVBX.H in place of it.
3. If you used AppStudio to put a Spread control on a dialog, open the .RC file as text in a text editor and make the following changes:

<b>Change:</b>	<b>To:</b>
CONTROL "SSPP20.VBX;Spreadsheet;	CONTROL "SSVBX25.VBX;vaSpread;

4. If you created the control at run time, make the following changes in your .CPP file:

<b>Change:</b>	<b>To:</b>
Create("SSPP20.VBX;Spreadsheet;" . . . )	Create("SSVBX25.VBX;vaSpread;" . . . )

5. In your project, where you loaded the VBX file, change the following reference:

<b>Change . . .</b>	<b>To . . .</b>
SSPP20.VBX	SSVBX25.VBX

6. If you have linked to the file SSPP.CPP, link instead to the following file as appropriate:

<b>If you are using . . .</b>	<b>Use . . .</b>
MFC	SSVBXMFC.CPP
OWL	SSVBXOWL.CPP

7. Rebuild your project and run it.



## Upgrading Existing Grid/VBX Version 1.0 Projects

If you have created applications using Grid/VBX version 1.0, you must upgrade your existing projects. Complete the steps provided for your development environment.

- [Upgrading Visual Basic Projects to Use the VBX Control](#)
- [Upgrading Visual Basic Projects to Use the OCX Controls](#)
- [Upgrading C++ Projects to Use the VBX Controls](#)

## Upgrading Visual Basic Projects to Use the VBX Control

If you formerly used Grid/VBX version 1.0 control and now want to use the Spread version 2.5 VBX control, complete the following steps.

### To upgrade existing Visual Basic projects for the VBX control

1. Open your existing project.
2. If the project used the file FPGRID.BAS, use the file SSVBX.BAS in place of it.
3. Save the project file and form files as text.
4. Edit the project file (.MAK) in a text editor to change the following references:  

<b>Change . . .</b>	<b>To . . .</b>
FPGRID10.VBX	SSVBX25.VBX
5. Edit the form files (.FRM) in a text editor to change all Grid class references as follows:  

<b>Change . . .</b>	<b>To . . .</b>
SpreadSheet	vaSpread
6. Load your project and verify your changes.

## Upgrading Visual Basic Projects to Use the OCX Controls

If you formerly used the Grid/VBX version 1.0 control and now want to use Spread version 2.5 OCX controls, complete the following steps.

### To upgrade existing Visual Basic projects for OCX controls

1. Open your existing project.
2. If the project used the file FPGRID.BAS, use the file SSOCX.BAS in place of it.
3. Save the project file and form files as text.
4. Edit the project file (.MAK) in a text editor to change the following references:  

<b>Change . . .</b>	<b>To . . .</b>
FPGRID10.VBX	SSVBX25.VBX
5. Edit the form files (.FRM) in a text editor to change all Spread class references as follows:  

<b>Change . . .</b>	<b>To . . .</b>
SpreadSheet	vaSpread
6. Load your project and verify your changes.
7. Save the project to rewrite the binary data in the .FRX file.
8. Start Visual Basic 4.0 (16-bit) or Visual Basic 4.0 (32-bit) and load the project file (.MAK).
9. For each program file (.FRM) containing a grid, update the event parameters as needed. See the file EVENTS.TXT for the parameter changes needed for using the events with the OCX control.
10. For each program file (.FRM) containing a grid, functions must be changed to methods. Compare the following code:

#### Visual Basic code using Grid/VBX 2.0

```
ret = SaveTabFile(Grid1, "\Test.txt")
fpSetText(Grid1, Col, Row, "Text")
```

#### Visual Basic code using the OCX control

```
ret = Grid1.SaveTabFile("\Test.txt")
Grid1.SetText Col, Row, "Text"
```

## Upgrading C++ Projects to Use the VBX Control

If you formerly used the Grid/VBX version 1.0 control and now want to use the Spread version 2.5 VBX control, complete the following steps.

### To upgrade existing C++ projects for the VBX control

1. Open your existing project.
2. If the project used the file FPGRID.H, use the file SSVBX.H in place of it.
3. If you used AppStudio to put a Spread control on a dialog, open the .RC file as text in a text editor and make the following changes:

<b>Change:</b>	<b>To:</b>
CONTROL "FPGRID10.VBX;SpreadSheet;	CONTROL "SSVBX25.VBX;vaSpread;

4. If you created the control at run time, make the following changes in your .CPP file:

<b>Change:</b>	<b>To:</b>
Create("FPGRID10.VBX;Spreadsheet;" . . . )	Create("SSVBX25.VBX;vaSpread;" . . . )

5. In your project, where you loaded the VBX file, change the following reference:

<b>Change . . .</b>	<b>To . . .</b>
FPGRID10.VBX	SSVBX25.VBX

6. If you have linked to the file FPGRID.CPP, link instead to the following file as appropriate:

<b>If you are using . . .</b>	<b>Use . . .</b>
MFC	SSVBXMFC.CPP
OWL	SSVBXOWL.CPP

7. Rebuild your project and run it.

## Documentation Conventions

Spread documentation uses the following conventions:

<b>Example</b>	<b>Description</b>
Example	Choose this item to display an example window.
C++	Text indicates the condition for the subsequent instructions or explanation. For example, the letters C++ in the left margin indicate that the subsequent information pertains only to the C++ programming language. Possible conditions include version of the control and development environment.
A:\SETUP	Words you need to type appear in this font.
SSVBX.BAS	Words in all capital letters indicate file and path names.
<i>Col, Row</i>	Italicized items in programming syntax are placeholders for information you supply.
<b>short, BOOL</b>	In programming syntax, bold, italicized arguments indicate data types, pointers, and user-defined types.

## Contacting FarPoint Technologies, Inc.

If you discover a problem with either the software or the accompanying documentation, or if you would like to share your thoughts about this product with FarPoint Technologies, please send us a completed [Comment Form](#).

Send us the Comment Form using one of the following methods, or if you call, please provide the information requested on the form to our technical support representative. For additional assistance, contact our Technical Support department.

### To contact FarPoint Technologies, Inc.

- **Call** our Technical Support department at (919) 460-1887.  
FarPoint Technologies' Technical Support department is available between the hours of 9:00 a.m. and 5:30 p.m. EST.
- **Fax** us at (919) 460-7606.
- Contact the FarPoint forum on **CompuServe** (type GO FARPOINT).
- **Write** us at:  
FarPoint Technologies, Inc.  
133 Southcenter Court  
Suite 1000  
Morrisville, NC 27560
- Send **e-mail** to us at [farpoint@fpoint.com](mailto:farpoint@fpoint.com).  
You can also contact FarPoint Technologies at
- Our **World Wide Web** site (<http://www.fpoint.com>)
- Our **FTP** site (<ftp.fpoint.com> and change to the directory [/fpoint.com/incoming](ftp.fpoint.com/incoming))

## Spread Comment Form

If you encounter a problem with this product, or if you would like to share your comments with FarPoint Technologies, please complete and send this form to FarPoint Technologies.

If you are reporting a problem, please include a detailed description of the problem along with the steps necessary to reproduce it. The preferred method is a sample application that reproduces the problem. If you have a sample, please upload it to our CompuServe forum (GO FARPOINT), or to our Internet address (fpsupport@fpoint.com or FTP to ftp.fpoint.com and change to the directory /fpoint.com/incoming). Be sure to include the name of the \*.ZIP file that you uploaded.

Today's date: \_\_\_\_\_ Customer name: \_\_\_\_\_  
Company name: \_\_\_\_\_ Phone number: (\_\_\_\_) \_\_\_\_\_  
Fax number: (\_\_\_\_) \_\_\_\_\_ E-mail address: \_\_\_\_\_  
\*Sample application file name: \_\_\_\_\_  
Version Number: \_\_\_\_\_ Control Date: \_\_\_\_\_  
Control Type (circle one or more): 16-bit DLL      32-bit DLL      VBX      16-bit OCX      32-bit OCX  
Development environment: \_\_\_\_\_ Version: \_\_\_\_\_  
(for example, Visual Basic, Visual C++, Borland C++)  
Description of the problem:

Steps necessary to reproduce the problem:

Comments/Documentation suggestions:

## Using OCX Controls

- [Overview](#)
- [Types of OCX Properties](#)
- [Setting Properties](#)
- [Spread Control Property Pages](#)
- [Using Events](#)
- [Summary of OCX Control Features](#)

## **Overview: OCX Controls**

OCX controls are OLE 2 custom controls. You can use the Spread OCX controls in any development environment that supports OCX controls.

OCX controls are designed to function more independently than VBX and DLL controls. That is, they are designed to be independent objects that you can easily use in many different types of containers and environments. Designed to present a consistent interface, OCX controls and their containers provide a unique means for interaction.

The following topics provide an overview of some OCX-specific concepts and features. Throughout the help, OCX-specific information has been included to further acquaint you with the Spread OCX controls and to provide instructions for their use.



## Types of OCX Properties

OCX controls can support four types of properties: stock, standard extender, custom, and ambient.

Stock properties are always provided by containers that support OCXs. Examples of stock properties are the [BackColor](#) and [Enabled](#) properties.

Standard extender properties can be, but are not necessarily, provided by containers that support OCXs. Examples of standard extender properties include the [Height](#) and [Left](#) properties.

Custom properties are properties added by the control's designer to support certain characteristics of the OCX control. For example, the [OperationMode](#) and [VirtualMode](#) properties are custom properties.

Ambient properties are properties that let the OCX control respond to characteristics of its container. For example, the `AmbientScaleUnit` property lets the OCX control use the same scale unit as its container. Note that containers need not support ambient properties.

The [property topics](#) provide information about the OCX type for the Spread control's properties. Each property that is an OCX stock or standard extender property is noted. If no information about the OCX type is provided for a given property, that property is a custom property.

## Setting Properties

OCX controls provide multiple avenues for viewing and setting control properties. You can view and set properties in tabs called property pages, which are organized by control characteristics. You can also view all the design-time properties available for the control in a list called a property browser. Finally, you can set properties in code at run time. The type of code you use and how you provide it depends on your container.

## Property Pages

Property pages are collections of tabs that present design-time properties for the OCX control. Each page contains one or more controls that let you set characteristics of the OCX control. Often, the controls on the page are grouped; for example, the Display property page contains a Scroll Bars group that groups all scroll bar characteristics together.

Usually, each control on a property page has a one-to-one correspondence to a property. For example, the Max Number of Rows box corresponds to the [MaxRows](#) property. However, sometimes a control represents a particular setting of a property.

Each property page contains standard buttons: OK, Cancel, Apply, and Help. Choose the OK button to close the property page window and to apply any changes you have made. Choose the Cancel button to close the window without applying any changes. Choose the Apply button to apply your changes without closing the window. Choose the Help button to access context-sensitive help about the current property page. Note that any time you change the active property page, the changes you have made on that page are applied to the control.

Stock properties such as [Text](#) and [BackColor](#) are often provided on property pages. Most design-time custom properties are provided as well. Standard extender properties are not provided on property pages because they are not necessarily supported by all containers.

The Spread OCX controls provide eight property pages: [General](#), [Display](#), [Op Mode](#), [Headers](#), [Edit Mode](#), [Virtual Mode](#), [Fonts](#), and [Colors](#).

## Property Browsers

Each container provides a property browser as a means for you to view an alphabetical list of the available design-time properties for the OCX control. For example, in Visual Basic the Properties window provides a list that lets you view and set properties. In addition, you can view the available settings for enumerated properties. However, different containers provide different browser interfaces.

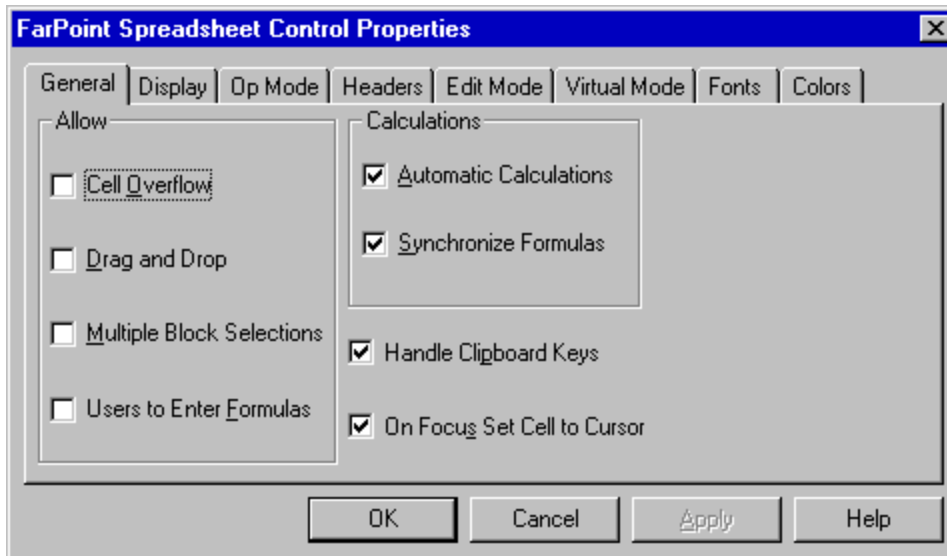
## Spread Control Property Pages

The Spread OCX controls provide eight property pages for customizing your interface.

- [General Property Page](#)
- [Display Property Page](#)
- [Op Mode Property Page](#)
- [Headers Property Page](#)
- [Edit Mode Property Page](#)
- [Virtual Mode Property Page](#)
- [Fonts Property Page](#)
- [Colors Property Page](#)

## General Property Page

The General property page lets you customize characteristics of your Spread control such as whether text can overflow to adjacent cells and whether the spreadsheet recalculates formulas when the contents of dependent cells change.

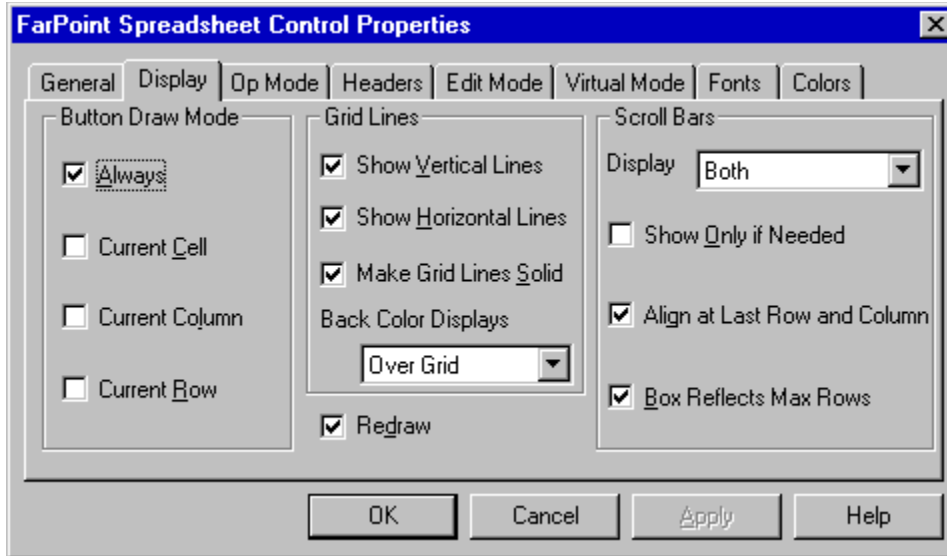


The following items are available on the General property page:

Item on Property Page	Description	Corresponding Property
Allow— Cell Overflow check box	Specifies whether text can overflow to adjacent empty cells	<a href="#">AllowCellOverflow</a>
Allow— Drag and Drop check box	Specifies whether the user can drag and drop cells	<a href="#">AllowDragDrop</a>
Allow— Multiple Block Selections check box	Specifies whether the user can select multiple, noncontiguous blocks of cells	<a href="#">AllowMultiBlocks</a>
Allow— Users to Enter Formulas check box	Specifies whether the user can type formulas into float or integer cells	<a href="#">AllowUserFormulas</a>
Calculations— Automatic Calculations check box	Specifies whether the spreadsheet recalculates formulas when the contents of dependent cells change	<a href="#">AutoCalc</a>
Calculations— Synchronize Formulas check box	Specifies whether the spreadsheet adjusts formulas when columns or rows are inserted or deleted or when blocks of cells are moved or swapped	<a href="#">FormulaSync</a>
Handle Clipboard Keys check box	Specifies whether the spreadsheet supports the cut, copy, and paste Clipboard shortcut keys	<a href="#">AutoClipboard</a>
On Focus Set Cell to Cursor check box	Specifies the location of the active cell when the user moves the focus to the spreadsheet with the mouse	<a href="#">MoveActiveOnFocus</a>

## Display Property Page

The Display property page lets you customize elements in the spreadsheet display, such as grid lines and scroll bars.

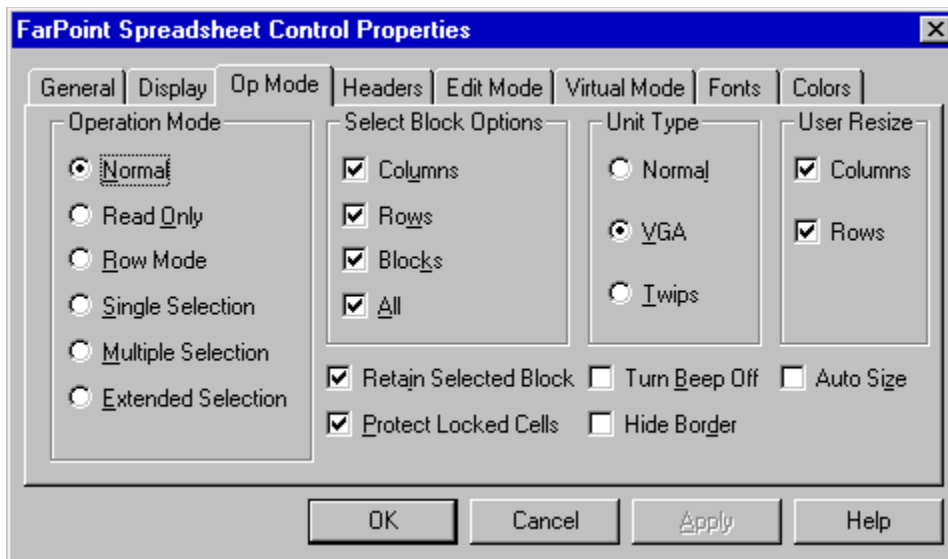


The following items are available on the Display property page:

Item on Property Page	Description	Corresponding Property
Button Draw Mode— Always check box Current Cell check box Current Column check box Current Row check box	Specify whether to display buttons in button and combo box cells	<a href="#">ButtonDrawMode</a>
Grid Lines— Show Vertical Lines check box	Specifies whether to display vertical grid lines	<a href="#">GridShowVert</a>
Grid Lines— Show Horizontal Lines check box	Specifies whether to display horizontal grid lines	<a href="#">GridShowHoriz</a>
Grid Lines— Make Grid Lines Solid check box	Specifies the type of grid lines displayed	<a href="#">GridSolid</a>
Grid Lines— Back Color Displays drop-down list box	Specifies whether the background colors of cells overlap the grid lines	<a href="#">BackColorStyle</a>
Redraw check box	Determines whether the application immediately redraws the Spread control after the user makes changes	<a href="#">ReDraw</a>
Scroll Bars— Display drop-down list box	Specifies whether and how the spreadsheet displays scroll bars	<a href="#">ScrollBars</a>
Scroll Bars— Show Only if Needed check box	Specifies whether to display scroll bars at all times or only when needed	<a href="#">ScrollBarExtMode</a>
Scroll Bars— Align at Last Row and Column check box	Specifies the alignment of the last row and column on the displayed portion of the spreadsheet	<a href="#">ScrollBarMaxAlign</a>
Scroll Bars— Box Reflects Max Rows check box	Specifies the maximum range of the scroll bars displayed in the spreadsheet	<a href="#">ScrollBarShowMax</a>

## Op Mode Property Page

The Op Mode property page lets you choose the operation mode and unit type for the spreadsheet, decide whether the user can select columns, rows, or blocks, and decide whether the user can resize columns and rows.



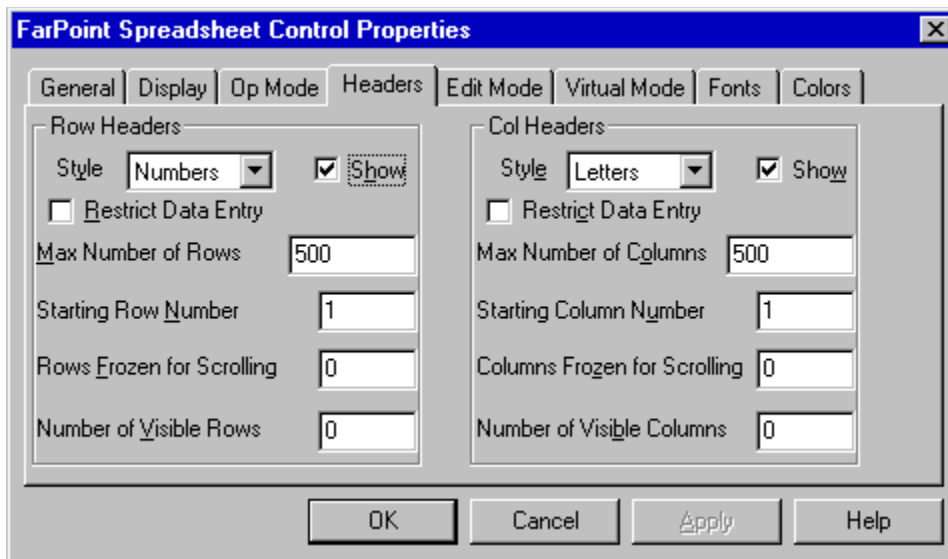
The following items are available on the Op Mode property page:

Item on Property Page	Description	Corresponding Property
Operation Mode— Normal option button Read Only option button Row Mode option button Single Selection option button Multiple Selection option button Extended Selection option button	Set the operation mode of the spreadsheet	<a href="#">OperationMode</a>
Select Block Options— Columns check box Rows check box Blocks check box All check box	Specify whether the user can select columns, rows, blocks of cells, all cells, or any combination of these	<a href="#">SelectBlockOptions</a>
Unit Type— Normal option button VGA option button Twips option button	Specify the units used for the column width and row height	<a href="#">UnitType</a>
User Resize— Columns check box Rows check box	Specify whether the user can resize columns, rows, or both	<a href="#">UserResize</a>
Retain Selected Block check box	Specifies whether a selected block of cells remains highlighted when the spreadsheet loses the focus	<a href="#">RetainSelBlock</a>
Protect Locked Cells check box	Specifies whether the user can edit cells marked as locked with the <a href="#">Lock</a> property	<a href="#">Protect</a>
Turn Beep Off check box	Specifies whether the spreadsheet sounds warning beeps	<a href="#">NoBeep</a>
Hide Border check box	Specifies whether to display a border on the bottom and right edges of the spreadsheet	<a href="#">NoBorder</a>
Auto Size check box	Specifies whether the spreadsheet is automatically sized to display only complete columns and rows	<a href="#">AutoSize</a>

## Headers Property Page

The Headers property page lets you customize the column header row and row header column, set the maximum number of columns and rows, and specify the number of visible columns and rows.



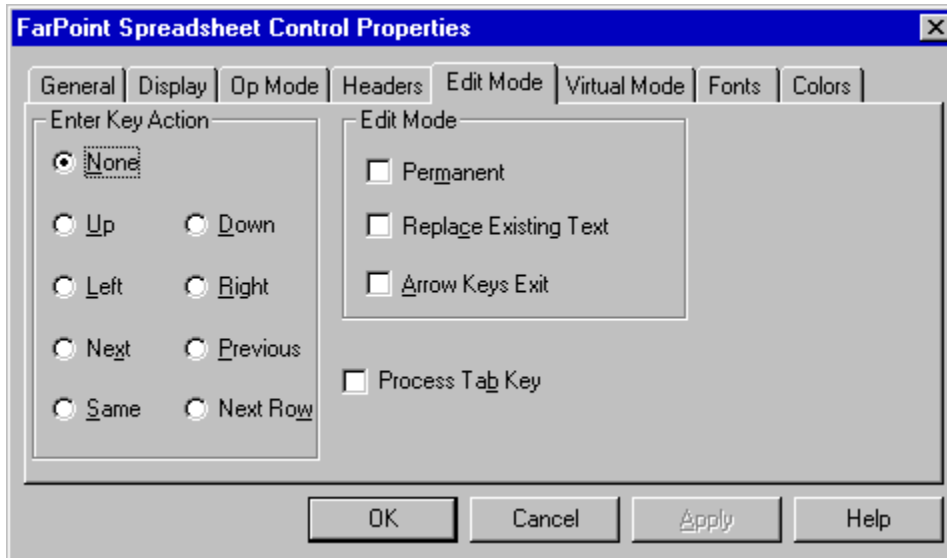


The following items are available on the Headers property page:

Item on Property Page	Description	Corresponding Property
Row Headers— Style drop-down list box	Specifies whether the row header column displays letters or numbers or is blank	<a href="#">RowHeaderDisplay</a>
Row Headers— Show check box	Specifies whether the spreadsheet displays the row header column	<a href="#">DisplayRowHeaders</a>
Row Headers— Restrict Data Entry check box	Specifies whether the user can type data in a row that is more than one row beyond the last row containing data	<a href="#">RestrictRows</a>
Row Headers— Max Number of Rows box	Sets the maximum number of usable rows	<a href="#">MaxRows</a>
Row Headers— Starting Row Number box	Specifies the row number of the first row to display	<a href="#">StartingRowNumber</a>
Row Headers— Rows Frozen for Scrolling box	Sets the number of nonscrolling, frozen rows	<a href="#">RowsFrozen</a>
Row Headers— Number of Visible Rows box	Sets the number of rows that are fully displayed	<a href="#">VisibleRows</a>
Col Headers— Style drop-down list box	Specifies whether the column header row displays letters or numbers or is blank	<a href="#">ColHeaderDisplay</a>
Col Headers— Show check box	Specifies whether the spreadsheet displays the column header row	<a href="#">DisplayColHeaders</a>
Col Headers— Restrict Data Entry check box	Specifies whether the user can type data in a column that is more than one column beyond the last column containing data	<a href="#">RestrictCols</a>
Col Headers— Max Number of Columns box	Sets the maximum number of usable columns	<a href="#">MaxCols</a>
Col Headers— Starting Column Number box	Specifies the column number of the first column to display	<a href="#">StartingColNumber</a>
Col Headers— Columns Frozen for Scrolling box	Sets the number of nonscrolling, frozen columns	<a href="#">ColsFrozen</a>
Col Headers— Number of Visible Columns box	Sets the number of columns that are fully displayed	<a href="#">VisibleCols</a>

## Edit Mode Property Page

The Edit Mode property page lets you customize keyboard functions during edit mode.

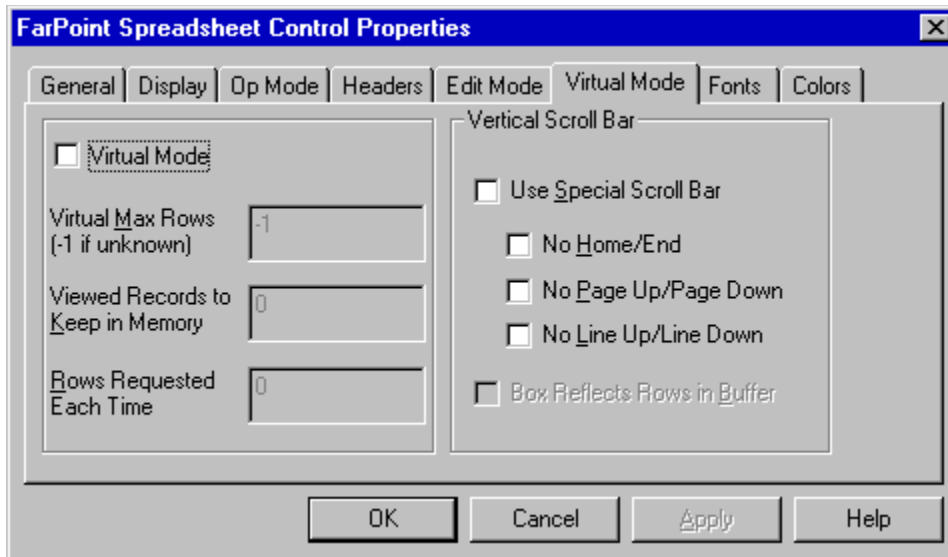


The following items are available on the Edit Mode property page:

Item on Property Page	Description	Corresponding Property
Enter Key Action— None option button Up option button Down option button Left option button Right option button Next option button Previous option button Same option button Next Row option button	Determine the action that occurs when the user presses the Enter key	<a href="#">EditEnterAction</a>
Edit Mode— Permanent check box	Specifies whether edit mode remains on when moving between cells	<a href="#">EditModePermanent</a>
Edit Mode— Replace Existing Text check box	Specifies whether cell text is replaced or appended during edit mode	<a href="#">EditModeReplace</a>
Edit Mode— Arrow Keys Exit check box	Specifies the function of the arrow keys	<a href="#">ArrowsExitEditMode</a>
Process Tab Key check box	Specifies whether the Tab key moves the focus either to the next cell in the spreadsheet or to the next control on the dialog	<a href="#">ProcessTab</a>

## Virtual Mode Property Page

The Virtual Mode property page lets you specify whether virtual mode is on or off and defines how the spreadsheet looks during virtual mode.

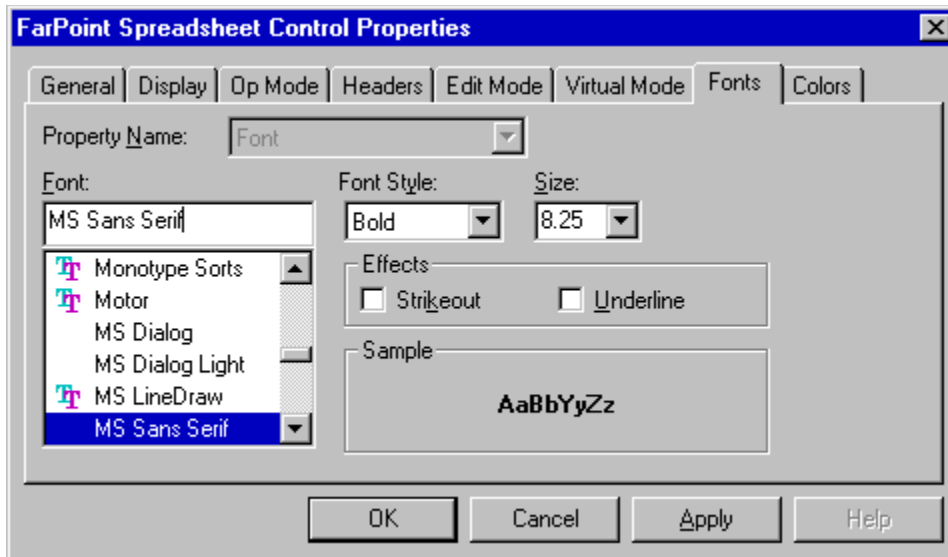


The following items are available on the Virtual Mode property page:

Item on Property Page	Description	Corresponding Property
Virtual Mode check box	Specifies whether virtual mode is on or off	<a href="#">VirtualMode</a>
Virtual Max Rows box	Sets the maximum number of rows allowed when virtual mode is on	<a href="#">VirtualMaxRows</a>
Viewed Records to Keep in Memory box	Sets the number of rows to keep in the virtual buffer when virtual mode is on	<a href="#">VirtualOverlap</a>
Rows Requested Each Time box	Sets the number of virtual rows the spreadsheet requests at a time	<a href="#">VirtualRows</a>
Vertical Scroll Bar— Use Special Scroll Bar check box	Specifies whether the spreadsheet uses a special vertical scroll bar	<a href="#">VScrollSpecial</a>
Vertical Scroll Bar— No Home/End check box No Page Up/Page Down check box No Line Up/Line Down check box	Specify the scroll arrows to display on the special vertical scroll bar	<a href="#">VScrollSpecialType</a>
Vertical Scroll Bar— Box Reflects Rows in Buffer check box	Specifies whether the scroll box reflects the number of rows in the virtual buffer rather than the maximum number of rows	<a href="#">VirtualScrollBuffer</a>

## Fonts Property Page

The Fonts property page lets you customize the fonts displayed in your Spread control. Note that OCX controls provide a [Font](#) property that lets you set characteristics for the text.

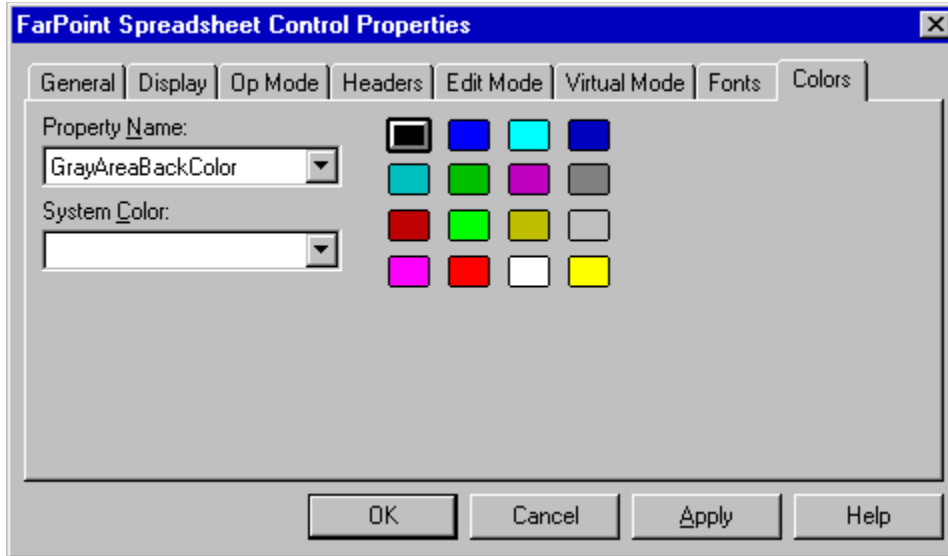


The following items are available on the Fonts property page:

Item on Property Page	Description	Corresponding Property
Property Name drop-down list box	Lists the available font properties for your control	
Font combo box	Lists the available fonts in your system	Font ( <a href="#">FontName</a> )
Font Style drop-down list box	Specifies the characteristics applied to the displayed font	Font ( <a href="#">FontBold</a> , <a href="#">FontItalic</a> )
Size drop-down combo box	Specifies the size in points of the displayed font	Font ( <a href="#">FontSize</a> )
Effects— Strikeout check box	Lets you display text with a line through it	Font ( <a href="#">FontStrikethru</a> )
Effects— Underline check box	Lets you display underlined text	Font ( <a href="#">FontUnderline</a> )
Sample	Displays sample of chosen font and font characteristics	

## Colors Property Page

The Colors property page lets you set the values for all the color properties. You can choose a color from the displayed color chart, or you can specify a system color from the System Color drop-down list box.



The following items are available on the Colors property page:

Item on Property Page	Description	Corresponding Property
Property Name drop-down list box	Lists all the available color properties you can set	(Individual properties in list)
System Color drop-down list box	Lets you specify a system color as the value for the specified color property	
Color chart	Lets you select one of the available colors as the value for the specified color property	

## Using Events

OCX controls can support three types of events: stock, standard extender, and custom.

Stock events are provided by all OCX containers. Examples of stock events include the [Click](#) event, the [DbClick](#) event, and the [KeyPress](#) event.

Standard extender events are events that OCX containers can, but do not necessarily, provide. Examples of standard extender events include the [DragDrop](#) event and the [GotFocus](#) event.

Custom events are created by the control's designer to support certain actions users perform on that control. For example, the Spread control includes the [ColWidthChange](#) event, which occurs when the user changes the column width using the mouse.

## Summary of OCX Control Features

- OCX controls are supported by any OLE-compliant containers that support OCX controls.
- OCX controls can support four types of properties: stock, standard extender, custom, and ambient.
- OCX controls provide several methods of interaction: property pages, a property browser, and run-time settings.
- The Spread OCX controls provide eight property pages: [General](#), [Display](#), [Op Mode](#), [Headers](#), [Edit Mode](#), [Virtual Mode](#), [Fonts](#), and [Colors](#).
- OCX controls can have three types of events: stock, standard extender, and custom.

## Customizing Columns and Rows

- [Inserting Columns and Rows](#)
- [Deleting Columns and Rows](#)
- [Sizing Columns and Rows](#)
- [Setting the Maximum Number of Columns and Rows](#)
- [Freezing Columns and Rows](#)
- [Hiding Columns and Rows](#)
- [Customizing the Column Header Row and Row Header Column](#)

## Inserting Columns and Rows

When you insert a column in a spreadsheet, the total number of columns does not increase. Current data in the selected column and in columns to the right shifts right and the last column in the spreadsheet, including data, is deleted. To ensure that the data in the far right column is not deleted, increase the maximum number of columns in the spreadsheet. For more information on changing the maximum number of columns, see [Setting the Maximum Number of Columns and Rows](#).

Similarly, when you insert a row in a spreadsheet, the total number of rows does not increase. Current data in the selected row and in rows below shifts down and the last row in the spreadsheet, including data, is deleted. To ensure that the data in the bottom row is not deleted, increase the maximum number of rows in the spreadsheet. For more information on changing the maximum number of rows, see [Setting the Maximum Number of Columns and Rows](#).

You can insert more than one column or row at a time. The number of columns or rows you select in block mode determines how many columns or rows are added.

When you insert columns or rows, you can specify whether formulas are adjusted. For more information, see [Automatically Recalculating and Updating Formulas](#).

▶ [OCX, VBX Instructions](#)



[Print](#)

[Copy](#)

[Close](#)

### To insert columns and rows

OCX, VBX

1. If you want to insert a column, at run time,
  - a. Specify the column before which the new column will be inserted with the [Col](#) property.
  - b. If you want to add more than one column, set the [Col2](#) property.
  - c. If you want to add more than one column, set the [BlockMode](#) property to True.
  - d. Set the [Action](#) property to 6 (Insert Col).
  - e. If you added more than one column, set the BlockMode property back to False.
2. If you want to insert a row, at run time,
  - a. Specify the row above which the new row will be inserted with the [Row](#) property.
  - b. If you want to add more than one row, set the [Row2](#) property.
  - c. If you want to add more than one row, set the [BlockMode](#) property to True.
  - d. Set the [Action](#) property to 7 (Insert Row).
  - e. If you added more than one row, set the BlockMode property back to False.

## Deleting Columns and Rows

When you delete a column from a spreadsheet, the total number of columns does not decrease. Current data in the selected column is deleted and data in the columns to the right of the deleted column shifts left. A new blank column appears as the far right column of the spreadsheet. If you want to decrease the number of columns in the spreadsheet, change the maximum number of columns immediately after deleting a column. For more information on changing the maximum number of columns, see [Setting the Maximum Number of Columns and Rows](#).

Similarly, when you delete a row from a spreadsheet, the total number of rows does not decrease. Current data in the selected row is deleted and data in rows below shifts up. A new blank row appears as the bottom row of the spreadsheet. If you want to decrease the number of rows in the spreadsheet, change the maximum number of rows immediately after deleting a row. For more information on changing the maximum number of rows, see [Setting the Maximum Number of Columns and Rows](#).

You can delete more than one column or row at a time. The number of columns or rows you select in block mode determines how many columns or rows are deleted.

When you delete columns or rows, you can specify whether formulas are adjusted. For more information, see [Automatically Recalculating and Updating Formulas](#).

▶ [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To delete columns and rows

OCX, VBX

1. If you want to delete a column, at run time,
  - a. Specify the column to be deleted with the [Col](#) property.
  - b. If you want to delete more than one column, set the [Col2](#) property.
  - c. If you want to delete more than one column, set the [BlockMode](#) property to True.
  - d. Set the [Action](#) property to 4 (Delete Col).
  - e. If you deleted more than one column, set the BlockMode property back to False.
2. If you want to delete a row, at run time,
  - a. Specify the row to be deleted with the [Row](#) property.
  - b. If you want to delete more than one row, set the [Row2](#) property.
  - c. If you want to delete more than one row, set the [BlockMode](#) property to True.
  - d. Set the [Action](#) property to 5 (Delete Row).
  - e. If you deleted more than one row, set the BlockMode property back to False.

## Sizing Columns and Rows

You can change the size of columns and rows by

- manually setting the column width and row height
- setting the column width or row height to fit the widest or tallest data
- using the mouse

You can prevent the user from resizing columns and rows with the mouse.

- [Setting the Width of Columns](#)
- [Setting the Height of Rows](#)
- [Resizing Columns and Rows to Fit Text](#)
- [Preventing Resizing of Columns and Rows with the Mouse](#)

## Setting the Width of Columns

You can set the width of a spreadsheet column or the width of the row header column to a specific value.

▶ [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To set the width of a column**

OCX, VBX

At run time, specify the width with the [ColWidth](#) property.

Specify the column number in the *Index* parameter of this property. To specify all columns, set the *Index* parameter and the [Col](#) property to  $-1$ . To specify the row header column, enter 0 as the index.

## Setting the Height of Rows

You can set the height of a spreadsheet row or the height of the column header row to a specific value.



[OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To set the height of a row**

OCX, VBX

At run time, specify the row height with the [RowHeight](#) property.

Specify the row in the *Index* parameter of this property. To specify all rows, set the *Index* parameter and the [Row](#) property to —

1. To specify the column header row, enter 0 as the index.



## Resizing Columns and Rows to Fit Text

You can resize a column to fit the widest text within that column and you can resize a row to fit the tallest text within that row.



[OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To resize columns and rows to fit text

OCX, VBX

1. If you want to resize a column to fit the widest text, at run time,
  - a. Set the [MaxTextColWidth](#) property to a variable.  
Specify the column in the *Index* parameter of this property. To specify the row header column, enter 0 as the index.
  - b. Set the [ColWidth](#) property to the variable defined in Step 1a.
2. If you want to resize a row to fit the tallest text, at run time,
  - a. Set the [MaxTextRowHeight](#) property to a variable.  
Specify the row in the *Index* parameter of this property. To specify the column header row, enter 0 as the index.
  - b. Set the [RowHeight](#) property to the variable defined in Step 2a.

## Preventing Resizing of Columns and Rows with the Mouse

By default, you can resize both columns and rows with the mouse. You can prevent resizing of columns, rows, or both. You can also prevent resizing of individual columns or rows.

- ▶ [OCX Instructions](#)
- ▶ [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To prevent resizing of columns and rows

OCX

1. On the [Op Mode](#) property page under User Resize,
  - a. If you want to prevent resizing of columns and rows, clear the Columns and Rows check boxes.
  - b. If you want to prevent resizing of columns only, clear the Columns check box.
  - c. If you want to prevent resizing of rows only, clear the Rows check box.
  - d. Choose the Apply button.
2. If you want to prevent resizing of an individual column, at run time,
  - a. Specify the column with the [Col](#) property.
  - b. Set the [UserResizeCol](#) property to 2 (Off).
3. If you want to prevent resizing of an individual row, at run time,
  - a. Specify the row with the [Row](#) property.
  - b. Set the [UserResizeRow](#) property to 2 (Off).

[Print](#)

[Copy](#)

[Close](#)

### To prevent resizing of columns and rows

VBX

1. If you want to prevent resizing of columns and rows, set the [UserResize](#) property to 0.
2. If you want to prevent resizing of columns only, set the UserResize property to 2 (Rows).
3. If you want to prevent resizing of rows only, set the UserResize property to 1 (Cols).
4. If you want to prevent resizing of an individual column, at run time,
  - a. Specify the column with the [Col](#) property.
  - b. Set the [UserResizeCol](#) property to 2 (Off).
5. If you want to prevent resizing of an individual row, at run time,
  - a. Specify the row with the [Row](#) property.
  - b. Set the [UserResizeRow](#) property to 2 (Off).

## Setting the Maximum Number of Columns and Rows

You can specify the maximum number of columns and rows in a spreadsheet. When you scroll beyond the maximum number of columns or rows, a gray area appears indicating no additional cells exist.

You might want to increase or decrease the maximum number of columns and rows when inserting or deleting columns and rows. For more information on inserting and deleting columns and rows, see [Inserting Columns and Rows](#) and [Deleting Columns and Rows](#).

- ▶ [OCX Instructions](#)
- ▶ [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To set the maximum number of columns and rows**

OCX

On the [Headers](#) property page,

1. Under Col Headers, type the number of columns in the Max Number of Columns box.
2. Under Row Headers, type the number of rows in the Max Number of Rows box.
3. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To set the maximum number of columns and rows**

VBX

1. Specify the number of columns with the [MaxCols](#) property.
2. Specify the number of rows with the [MaxRows](#) property.



## Freezing Columns and Rows

You can freeze (make nonscrollable) any number of columns and rows. The frozen columns are always the far left columns. The frozen rows are always the top rows.

**Note** Do not freeze rows in spreadsheets that are using virtual mode.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To freeze columns and rows**

OCX

On the [Headers](#) property page,

1. Under Col Headers, type the number of columns in the Columns Frozen for Scrolling box.
2. Under Row Headers, type the number of rows in the Rows Frozen for Scrolling box.
3. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To freeze columns and rows**

VBX

1. Specify the number of columns with the [ColsFrozen](#) property.
2. Specify the number of rows with the [RowsFrozen](#) property.

## Hiding Columns and Rows

You can hide a column or row in a spreadsheet. You can also hide the column header row and row header column.

— [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To hide a column or row

OCX, VBX

1. If you want to hide a column, at run time,
  - a. Specify the column to hide with the [Col](#) property.
  - b. Set the [ColHidden](#) property to True.
2. If you want to hide a row, at run time,
  - a. Specify the row to hide with the [Row](#) property.
  - b. Set the [RowHidden](#) property to True.

# Customizing the Column Header Row and Row Header Column

You can define the information displayed in the spreadsheet column header row and row header column. You can also choose not to display these headers.

row header column

column header row

	A	B	C
1			
2			
3			
4			

- [Specifying Header Content](#)
- [Hiding the Column Header Row and the Row Header Column](#)

## Specifying Header Content

The column header row and the row header column can contain sequential letters, sequential numbers, or custom text.

- [Specifying the Header Content Type](#)
- [Specifying Header Text](#)
- [Changing Column and Row Numbering](#)

## Specifying the Header Content Type

By default, the column header row displays letters and the row header column displays numbers. You can customize the column header row to display numbers or the row header column to display letters. You can also customize the headers to appear blank.

- [OCX Instructions](#)
- [VBX Instructions](#)



[Print](#)

[Copy](#)

[Close](#)

**To specify the header content type**

OCX

On the [Headers](#) property page,

1. Under Col Headers, select the column header type from the Style drop-down list box.
2. Under Row Headers, select the row header type from the Style drop-down list box.
3. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To specify the header content type**

VBX

1. Specify the column header type with the [ColHeaderDisplay](#) property.
2. Specify the row header type with the [RowHeaderDisplay](#) property.

## Specifying Header Text

You can display any text in the column header row and the row header column (instead of displaying sequential numbers or letters).

- [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To specify header text

OCX, VBX

1. If you want to enter text in the column header row, at run time,
  - a. Set the [Row](#) property to 0.
  - b. Select a column or a column range with the [Col](#) and [Col2](#) properties.
  - c. Specify the text with the [Text](#) property.
2. If you want to enter text in the row header column, at run time,
  - a. Set the [Col](#) property to 0.
  - b. Select a row or a row range with the [Row](#) and [Row2](#) properties.
  - c. Specify the text with the [Text](#) property.

## Changing Column and Row Numbering

If the column header row and the row header column display letters or numbers, you can define the starting letter or number. The starting letter or number you specify is used only for display purposes and has no effect on the actual row and column coordinates.

**Note** This value is expressed as an integer. For example, if the column header type is letter and you set the starting column letter to 10, the first column header begins with 'J'.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To change column and row numbering**

OCX

On the [Headers](#) property page,

1. Under Col Headers, type the starting column letter or number in the Starting Column Number box.
2. Under Row Headers, type the starting row letter or number in the Starting Row Number box.
3. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To change column and row numbering**

VBX

1. Specify the starting column letter or number with the [StartingColNumber](#) property.
2. Specify the starting row letter or number with the [StartingRowNumber](#) property.

## Hiding the Column Header Row and the Row Header Column

By default, the Spread control displays the column header row and the row header column. You can hide the column header row, the row header column, or both.

**Note** If your spreadsheet is in virtual mode, the row header column might not display the correct numbers for the rows. You can hide the row header column, or you can set other characteristics to hide row header column text or display the correct numbers. For more information, see [Customizing Virtual Mode](#).

- [OCX Instructions](#)
- [VBX Instructions](#)



[Print](#)

[Copy](#)

[Close](#)

**To hide the column header row and row header column**

OCX

On the [Headers](#) property page,

1. If you want to hide the column header row, under Col Headers, clear the Show check box.
2. If you want to hide the row header column, under Row Headers, clear the Show check box.
3. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To hide the column header row and row header column**

VBX

1. If you want to hide the column header row, set the [DisplayColHeaders](#) property to False.
2. If you want to hide the row header column, set [DisplayRowHeaders](#) property to False.

## Customizing the Spreadsheet Display

- [Hiding Spreadsheet Borders](#)
- [Setting Spreadsheet Measurement Units](#)
- [Customizing Grid Lines](#)
- [Specifying the Cursor Type](#)
- [Displaying Scroll Bars](#)
- [Limiting the Scroll Box Range](#)
- [Designating the Leftmost Column to Display](#)
- [Designating the Topmost Row to Display](#)
- [Displaying a Specified Number of Columns and Rows](#)
- [Specifying Where the Last Column and Row Stop Scrolling](#)
- [Disabling the Spreadsheet Beep](#)
- [Preventing the Spreadsheet from Flickering](#)
- [Displaying Selections When the Spreadsheet Does Not Have the Focus](#)
- [Using the Spreadsheet as a List Box](#)
- [Specifying When Buttons Appear](#)
- [Setting the Way Arrow Keys Work](#)
- [Customizing Colors](#)

## Hiding Spreadsheet Borders

You can hide the border that displays on the bottom and right edges of the spreadsheet.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To hide the spreadsheet border**

OCX

On the [Op Mode](#) property page,

1. Select the Hide Border check box.
2. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To hide the spreadsheet border**

VBX

Set the [NoBorder](#) property to True.

## Setting Spreadsheet Measurement Units

The column width of the spreadsheet is specified in internal units. This value is calculated as 1 unit being equal to the width of one character of the Windows system fixed font.

The row height is based on the current font. If the font is changed, the height of the row will change to the new size of the font. For more information on changing the font, see [Setting the Font](#).

You can change the unit type to twips or to a constant value provided by the Spread control.

**Note** You should set the unit type before using any other spreadsheet property or function. Failure to do so will produce unexpected results.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To set the spreadsheet measurement units**

OCX

On the [Op Mode](#) property page under Unit Type,

1. Select one of the measurement option buttons.
2. Choose the Apply button.



[Print](#)

[Copy](#)

[Close](#)

**To set the spreadsheet measurement units**

VBX

Set the [UnitType](#) property.

## Customizing Grid Lines

You can choose to show the horizontal grid lines, the vertical grid lines, or both. You can have the grid lines display as solid or dotted lines and you can specify the color of the grid lines.

By default, the background color of a cell overlaps (hides) the right and bottom sides of the cell's grid lines. You can specify that the complete grid lines display or that only the horizontal or vertical lines display.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To customize spreadsheet grid lines

OCX

1. On the [Display](#) property page under Grid Lines,
  - a. To hide horizontal grid lines, clear the Show Horizontal Lines check box.
  - b. To hide vertical grid lines, clear the Show Vertical Lines check box.
  - c. To display grid lines as dotted, clear the Make Grid Lines Solid check box.
  - d. To display the grid lines over the background color, select Under Grid, Under Horiz Grid, or Under Vert Grid from the Back Color Displays drop-down list box.
  - e. Choose the Apply button.
2. If you want to change the color of the grid lines, on the [Colors](#) property page,
  - a. Select the [GridColor](#) property from the Property Name drop-down list box.
  - b. Specify the color by selecting one of the displayed colors or by selecting one of the system colors from the System Color drop-down list box.
  - c. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

### To customize spreadsheet grid lines

VBX

1. If you want to hide the horizontal grid lines, set the [GridShowHoriz](#) property to False.
2. If you want to hide the vertical grid lines, set the [GridShowVert](#) property to False.
3. If you want to display the grid lines as dotted, set the [GridSolid](#) property to False.
4. If you want to display the grid lines over the background color, set the [BackColorStyle](#) property to 1 (Under Grid), 2 (Over Horizontal Grid Only), or 3 (Over Vertical Grid Only).
5. Change the color of the grid lines with the [GridColor](#) property.

## Specifying the Cursor Type

You can specify a cursor type for different areas of the spreadsheet. You can define a custom icon to display as the mouse pointer or you can choose one of four other cursor types (crosshair, arrow, column resize, or row resize).

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To specify the cursor type and area

At run time,

1. If you want to specify a user-defined icon, specify the icon with the [CursorIcon](#) property.
2. Specify the area of the spreadsheet for which you want the cursor with the [CursorType](#) property.
3. If you set the CursorIcon property in Step 1, set the [CursorStyle](#) property to 0 (User Defined). Otherwise, set the CursorStyle property to value other than 0 (User Defined) to specify the cursor for the area you set in Step 2.
4. Repeat Steps 1—3 for each area of the spreadsheet for which you want to specify a cursor.

[Print](#)

[Copy](#)

[Close](#)

**To specify the cursor type and area**

At run time,

1. Specify the area of the spreadsheet for which you want the cursor with the [CursorType](#) property.
2. Specify the cursor for the area you set in Step 1 with the [CursorStyle](#) property.
3. Repeat Steps 1—2 for each area of the spreadsheet for which you want to specify a cursor

## Displaying Scroll Bars

You can specify whether scroll bars display for your spreadsheet. If you decide to display scroll bars, you can designate when the scroll bars display.

- [Specifying When Scroll Bars Are Displayed](#)
- [Specifying Whether Scroll Bars Are Displayed](#)



## Specifying When Scroll Bars Are Displayed

You can control whether scroll bars display only when the maximum number of columns and rows exceed the Spread control boundaries, or display at all times.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To specify when scroll bars are displayed**

OCX

On the [Display](#) property page under Scroll Bars,

1. If you want to hide the scroll bars when the maximum number of columns or rows fit in the control boundaries, select the Show Only if Needed check box.
2. If you want to display scroll bars at all times, clear the Show Only if Needed check box.
3. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To specify when scroll bars are displayed**

VBX

1. If you want to hide the scroll bars when the maximum number of columns or rows fit in the control boundaries, set the [ScrollBarExtMode](#) property to True.
2. If you want to display scroll bars at all times, set the ScrollBarExtMode property to False.

## Specifying Whether Scroll Bars Are Displayed

You can display no scroll bars or you can display the horizontal scroll bar, the vertical scroll bar, or both.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To specify whether scroll bars are displayed**

OCX

On the [Display](#) property page under Scroll Bars,

1. If you want to hide the scroll bars, select None from the Display drop-down list box.
2. If you want to display the scroll bars, select Horizontal, Vertical, or Both from the Display drop-down list box.
3. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To specify whether scroll bars are displayed**

VBX

1. If you want to hide the scroll bars, set the [ScrollBars](#) property to 0 (None).
2. If you want to display the scroll bars, set the ScrollBars property to either 1 (Horizontal), 2 (Vertical), or 3 (Both).

## Limiting the Scroll Box Range

By default, the scroll box position reflects the maximum number of columns and rows in the spreadsheet. For example, if the maximum number of rows is 200, the range for the vertical scroll box is 1 to 200. In other words, if you drag the scroll box to the bottom scroll arrow, row 200 (the last row) will display at the bottom of the spreadsheet.

You can limit the scroll box range to the number of columns or rows that contain data by setting the [ScrollBarShowMax](#) property to False. With this setting, the range of the horizontal scroll box is 1 to either 8 or the [DataColCnt](#) property value, whichever is greater. For the vertical scroll box, the range is 1 to either 20 or the [DataRowCnt](#) property value, whichever is greater.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To limit the scroll box range**

OCX

On the [Display](#) property page under Scroll Bars,

1. Clear the Box Reflects Max Rows check box.
2. Choose the Apply button.



[Print](#)

[Copy](#)

[Close](#)

**To limit the scroll box range**

VBX

Set the [ScrollBarShowMax](#) property to False.

## Designating the Leftmost Column to Display

You can set a specific column to be the far left column in the viewing area.

— [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To designate the leftmost column to display**

OCX, VBX

At run time, set the [LeftCol](#) property.

## Designating the Topmost Row to Display

You can set a specific row as the top scrollable row in the viewing area.

- [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To designate the topmost row to display**

OCX, VBX

At run time, set the [TopRow](#) property.

## Displaying a Specified Number of Columns and Rows

By default, the number of columns and rows that display in a spreadsheet is equal to the maximum number of usable columns and rows. You can define a specific number of columns and rows that are fully displayed in a spreadsheet.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To display a specified number of columns and rows**

OCX

1. On the [Headers](#) property page,
  - a. Under Col Headers, type the number of columns in the Number of Visible Columns box.
  - b. Under Row Headers, type the number of rows in the Number of Visible Rows box.
  - c. Choose the Apply button.
2. If you want to prevent partial display of columns and rows, on the [Op Mode](#) property page,
  - a. Select the Auto Size check box.
  - b. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To display a specified number of columns and rows**

VBX

1. Specify the number of columns with the [VisibleCols](#) property.
2. Specify the number of rows with the [VisibleRows](#) property.
3. If you want to prevent partial display of columns and rows, set the [AutoSize](#) property to True.



## Specifying Where the Last Column and Row Stop Scrolling

By default, the last column and row stop scrolling at the lower-right corner of the spreadsheet. You can specify that the last column and row stop scrolling at the upper-left corner of the spreadsheet.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To specify where the last column and row stop scrolling**

OCX

On the [Display](#) property page under Scroll Bars,

1. If you want the last column and row to stop scrolling at the lower-right corner, select the Align at Last Row and Column check box.
2. If you want the last column and row to scroll to the upper-left corner, clear the Align at Last Row and Column check box.
3. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To specify where the last column and row stop scrolling**

VBX

1. If you want the last column and row to stop scrolling at the lower-right corner, set the [ScrollBarMaxAlign](#) property to True.
2. If you want the last column and row to scroll to the upper-left corner, set the ScrollBarMaxAlign property to False.

## Disabling the Spreadsheet Beep

You can prevent the user from typing data into a column that is more than one column beyond the last column containing data. This ensures that the user types data one column at a time. The spreadsheet beeps when the user attempts to type data beyond that point. This is also true for rows.

You can disable the spreadsheet beep.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To disable the spreadsheet beep**

OCX

On the [Op Mode](#) property page,

1. Select the Turn Beep Off check box.
2. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To disable the spreadsheet beep**

VBX

Set the [NoBeep](#) property to True.

## Preventing the Spreadsheet from Flickering

By default, when you set a property or function that changes the spreadsheet's appearance or data, the spreadsheet automatically refreshes itself. However, when you make multiple changes or add large amounts of data, the spreadsheet appears as if it is flickering because the spreadsheet is redrawing itself multiple times. You can prevent these multiple spreadsheet redraws and flickering. This will also make your application run faster.

— [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To prevent spreadsheet flickering**

OCX, VBX

1. Set the [ReDraw](#) property to False.
2. Perform the necessary operations.
3. Set the ReDraw property to True.



## Displaying Selections When the Spreadsheet Does Not Have the Focus

You can specify whether a selected block of cells retains the focus rectangle when the spreadsheet does not have the focus. If the selected block does not display the focus rectangle when the spreadsheet does not have the focus, the focus rectangle will be restored when the spreadsheet regains the focus.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To display selections when the spreadsheet does not have the focus**

OCX

On the [Op Mode](#) property page,

1. If you want to retain the focus rectangle when the spreadsheet does not have the focus, select the Retain Selected Block check box.
2. If you want to remove the focus rectangle when the spreadsheet does not have the focus, clear the Retain Selected Block check box.
3. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To display selections when the spreadsheet does not have the focus**

VBX

1. If you want to retain the focus rectangle when the spreadsheet does not have the focus, set the [RetainSelBlock](#) property to True.
2. If you want to remove the focus rectangle when the spreadsheet does not have the focus, set the RetainSelBlock property to False.

## Using the Spreadsheet as a List Box

You can set the spreadsheet to operate like a single-selection, multiple-selection, or extended-selection list box.

For more information about spreadsheet operation modes, see [Specifying the Operation Mode](#).

**Note** Do not set the spreadsheet to use multiple- or extended-selection operation mode if it is using virtual mode.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To use the spreadsheet as a list box**

OCX

On the [Op Mode](#) property page under Operation Mode,

1. If you want the spreadsheet to operate like a single-selection list box, select the Single Selection option button.
2. If you want the spreadsheet to operate like a multiple-selection list box, select the Multiple Selection option button.
3. If you want the spreadsheet to operate like an extended-selection list box, select the Extended Selection option button.
4. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To use the spreadsheet as a list box**

VBX

1. If you want the spreadsheet to operate like a single-selection list box, set the [OperationMode](#) property to 3 (Single Select).
2. If you want the spreadsheet to operate like a multiple-selection list box, set the OperationMode property to 4 (Multi Select).
3. If you want the spreadsheet to operate like an extended-selection list box, set the OperationMode property to 5 (Extended Select).

## Specifying When Buttons Appear

When you use combo boxes or buttons in cells, you can specify when the button (in a button cell) or combo box arrow (in a combo box cell) displays: always, only in the current cell, only in the current column, or only in the current row. You might want to hide buttons to keep your spreadsheet from looking cluttered. For example, if your spreadsheet has an entire column of combo box cells, you might want to display the button only in the current cell.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To specify when buttons appear**

OCX

On the [Display](#) property page under Button Draw Mode,

1. Select the appropriate check box.
2. Choose the Apply button.



[Print](#)

[Copy](#)

[Close](#)

**To specify when buttons appear**

VBX

Set the [ButtonDrawMode](#) property.

## Setting the Way Arrow Keys Work

When working in edit mode, you can set the arrow keys to function in one of two ways:

- Pressing an arrow key causes the cell to exit edit mode and move the input focus in the appropriate direction.
- Pressing an arrow key moves the cursor within the cell and the cell remains in edit mode.
- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To set the way arrow keys work**

OCX

On the [Edit Mode](#) property page under Edit Mode,

1. If you want to use the arrow keys to exit edit mode, select the Arrow Keys Exit check box.
2. If you want to use the arrow keys to move within the cell, clear the Arrow Keys Exit check box.
3. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

### To set the way arrow keys work

VBX

1. If you want to use the arrow keys to exit edit mode, set the [ArrowsExitEditMode](#) property to True.
2. If you want to use the arrow keys to move within the cell, set the ArrowsExitEditMode property to False.  
To exit edit mode and move out of the current cell, press Ctrl + an arrow key.

## Customizing Colors

You can specify the background, border, text, gray area, and grid line colors in a spreadsheet.

- [Setting the Background and Text Colors](#)
- [Setting the Border Color](#)
- [Setting the Gray Area Color](#)
- [Setting the Grid Line Color](#)

## Setting the Background and Text Colors

You can customize the background and text colors of selected, locked, and button cells, and row and column headers.

- [Cells - Background and Text Colors](#)
- [Button Cells - Background and Text Colors](#)
- [Locked Cells - Background and Text Colors](#)
- [Column Header Row and Row Header Column - Background and Text Colors](#)

## Cells - Background and Text Colors

You can change the background and text colors of a cell, a column, a row, a block of cells, or the entire spreadsheet.

By default, the background color of a cell overlaps the right and bottom sides of the cell's grid lines. You can specify that the complete grid lines display. For more information, see [Customizing Grid Lines](#).

If you want to specify the background and text colors for button cells, see [Button Cells - Background and Text Colors](#).

— [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To set the background and text colors of cells

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. Specify the background color with the [BackColor](#) property.
4. Specify the text color with the [ForeColor](#) property.
5. If you selected a block of cells, set the BlockMode property back to False.



## Button Cells - Background and Text Colors

By default, the background color of a button cell is the Windows system button face color and the text color is black. You can customize the background and text colors.

- [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To set the background and text colors of button cells

OCX, VBX

At run time,

1. Specify the background color with the [TypeButtonColor](#) property.
2. Specify the text color with the [TypeButtonTextColor](#) property.

## Locked Cells - Background and Text Colors

By default, the background and text colors of locked cells are the same as the background and text colors of unlocked or normal cells. You can change the background and text colors of locked cells.

- [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To set the background and text colors of locked cells**

OCX, VBX

At run time,

1. Specify the background color with the [LockBackColor](#) property.
2. Specify the text color with the [LockForeColor](#) property.

## Column Header Row and Row Header Column - Background and Text Colors

You can set the colors of the background shadow and text in the column header row and the row header column.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To set the background and text colors for the column header row and the row header column**

OCX

On the [Colors](#) property page,

1. Select either of the following properties from the Property Name drop-down list box:
  - Set the background shadow color with the [ShadowColor](#) property.
  - Set the text color with the [ShadowText](#) property.
2. Specify the color of the selected property by selecting one of the displayed colors or by selecting one of the system colors from the System Color drop-down list box.
3. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To set the background and text colors for the column header row and the row header column**

VBX

1. Specify the background shadow color with the [ShadowColor](#) property.
2. Specify the text color with the [ShadowText](#) property.

## Setting the Border Color

You can customize the border color of selected and button cells, and the border color of the column header row and the row header column.

- [Cells - Border Color](#)
- [Button Cells - Border Color](#)
- [Column Header Row and Row Header Column - Border Color](#)



## Cells - Border Color

You can customize the border color of a cell, a column, a row, a block of cells, or the entire spreadsheet.

If you want to specify the colors of the border and the dark and light highlight areas for button cells, see [Button Cells - Border Color](#).

— [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To set the border color of cells

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. Specify the border color of the selected cells with the [CellBorderColor](#) property.
4. Set the [Action](#) property to 16 (Set Border).
5. If you selected a block of cells, set the BlockMode property back to False.

## Button Cells - Border Color

You can change the colors of a button cell border and the dark and light highlight areas that create a three-dimensional effect around the button cell.

By default, the button cell border is black. The default highlight area colors depend on your Windows system button highlight and shadow colors.

— [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### **To set the button cell border color**

OCX, VBX

At run time,

1. If you want to specify the border color, set the [TypeButtonBorderColor](#) property.
2. If you want to specify the dark highlight color, set the [TypeButtonDarkColor](#) property.
3. If you want to specify the light highlight color, set the [TypeButtonLightColor](#) property.

## Column Header Row and Row Header Column - Border Color

By default, the color of the bottom and right borders of the column header row and the row header column is dark gray. You can change the bottom and right border color of the column header row and the row header column.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To customize the bottom and right border color of the column header row and the row header column**

OCX

On the [Colors](#) property page,

1. Select the [ShadowDark](#) property from the Property Name drop-down list box.
2. Specify the color by selecting one of the displayed colors or by selecting one of the system colors from the System Color drop-down list box.
3. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

To customize the bottom and right border color of the column header row and the row header column  
VBX

Set the [ShadowDark](#) property.

## Setting the Gray Area Color

If you scroll beyond the maximum columns and rows in a spreadsheet, a gray area appears indicating no more cells exist. By default, the color of this gray area is light gray. You can customize the gray area color.

- [OCX Instructions](#)
- [VBX Instructions](#)



[Print](#)

[Copy](#)

[Close](#)

**To set the gray area color**

OCX

On the [Colors](#) property page,

1. Select the [GrayAreaBackColor](#) property from the Property Name drop-down list box.
2. Specify the color by selecting one of the displayed colors or by selecting one of the system colors from the System Color drop-down list box.
3. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To set the gray area color**

VBX

Set the [GrayAreaBackColor](#) property.

## Setting the Grid Line Color

The default grid line color is light gray. You can change the color of the grid lines.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To set the grid line color**

OCX

On the [Colors](#) property page,

1. Select the [GridColor](#) property from the Property Name drop-down list box.
2. Specify the color by selecting one of the displayed colors or by selecting one of the system colors from the System Color drop-down list box.
3. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To set the grid line color**

VBX

Set the [GridColor](#) property.

## Working with Cells

- [Designating the Cell Type](#)
- [Customizing Cell Borders](#)
- [Locking Cells](#)
- [Resizing Cells](#)
- [Allowing Cells to Overflow](#)
- [Allowing the User to Drag and Drop Cells](#)
- [Selecting Cells](#)
- [Working with a Block of Cells](#)
- [Setting the Font](#)
- [Using Edit Mode](#)
- [Navigating within Cells](#)

## Designating the Cell Type

Cell types define the type of information that appears in a cell. Examples of cell types include button, combo box, date, and integer.

You can specify the cell type for rows, columns, individual cells, or selected groups of cells in a spreadsheet. The following table summarizes the cell types.

<b>Cell Type</b>	<b>Description</b>
Button	Displays a push button in a cell (The button cell can contain text, graphics, or both.)
Check Box	Displays a check box in a cell
Combo Box	Displays a combo box in a cell and provides a list of items for display in the combo box
Date	Displays or allows editing of a formatted date value in a cell
Edit	Displays or allows editing of text in a cell
Float	Displays or allows editing of a formatted floating-point value in a cell (This cell type is usually used to display currency values.)
Integer	Displays or allows editing of an integer value in a cell (If you are using Visual Basic, this cell type handles long integers.)
Owner-drawn	Displays a user-defined cell type
PIC	Displays or allows editing of a formatted PIC (mask) in a cell
Picture	Displays a picture in a cell
Static Text	Displays noneditable, nonscrolling text in a cell
Time	Displays or allows editing of a formatted time value in a cell

- [Creating a button cell](#)
- [Creating a check box cell](#)
- [Creating a combo box cell](#)
- [Creating a date cell](#)
- [Creating an edit cell](#)
- [Creating a float cell](#)
- [Creating an integer cell](#)
- [Creating an owner-drawn cell](#)
- [Creating a PIC cell](#)
- [Creating a picture cell](#)
- [Creating a static text cell](#)
- [Creating a time cell](#)

[Print](#)

[Copy](#)

[Close](#)

### To create a button cell

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. Set the [CellType](#) property to 7 (Button).
4. Use the following properties to customize the button cell:

[TypeButtonAlign](#)

[TypeButtonBorderColor](#)

[TypeButtonColor](#)

[TypeButtonDarkColor](#)

[TypeButtonLightColor](#)

[TypeButtonShadowSize](#)

[TypeButtonPicture](#)

[TypeButtonText](#)

[TypeButtonPictureDown](#)

[TypeButtonTextColor](#)

[TypeButtonType](#)

5. If you selected a block of cells, set the BlockMode property back to False.



[Print](#)

[Copy](#)

[Close](#)

### To create a check box cell

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. Set the [CellType](#) property to 10 (Check Box).
4. Use the following properties to customize the check box cell:

[TypeCheckCenter](#)

[TypeCheckPicture](#)

[TypeCheckText](#)

[TypeCheckTextAlign](#)

[TypeCheckType](#)

5. If you selected a block of cells, set the BlockMode property back to False.

[Print](#)

[Copy](#)

[Close](#)

### To create a combo box cell

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. Set the [CellType](#) property to 8 (Combo Box).
4. Use the following properties to customize the combo box cell:  
[TypeComboBoxEditable](#)  
[TypeComboBoxList](#)
5. If you selected a block of cells, set the BlockMode property back to False.

[Print](#)

[Copy](#)

[Close](#)

### To create a date cell

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. Set the [CellType](#) property to 0 (Date).
4. Use the following properties to customize the date cell:

[TypeDateCentury](#)

[TypeDateFormat](#)

[TypeHAlign](#)

[TypeDateMax](#)

[TypeDateMin](#)

[TypeDateSeparator](#)

[TypeSpin](#)

5. If you selected a block of cells, set the [BlockMode](#) property back to False.

[Print](#)

[Copy](#)

[Close](#)

### To create an edit cell

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. Set the [CellType](#) property to 1 (Edit).
4. Use the following properties to customize the edit cell:

[TypeEditCharCase](#)

[TypeEditCharSet](#)

[TypeEditLen](#)

[TypeHAlign](#)

[TypeEditMultiLine](#)

[TypeEditPassword](#)

5. If you selected a block of cells, set the BlockMode property back to False.

[Print](#)

[Copy](#)

[Close](#)

### To create a float cell

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. Set the [CellType](#) property to 2 (Float).
4. Use the following properties to customize the float cell:

[TypeFloatMoney](#)

[TypeFloatCurrencyChar](#)

[TypeHAlign](#)

[TypeFloatDecimalChar](#)

[TypeFloatSeparator](#)

[TypeFloatSepChar](#)

[TypeFloatDecimalPlaces](#)

[TypeFloatMax](#)

[TypeFloatMin](#)

5. If you selected a block of cells, set the BlockMode property back to False.

[Print](#)

[Copy](#)

[Close](#)

### To create an integer cell

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. Set the [CellType](#) property to 3 (Integer).
4. Use the following properties to customize the integer cell:

[TypeIntegerMax](#)

[TypeIntegerMin](#)

[TypeSpin](#)

[TypeHAlign](#)

[TypeIntegerSpinInc](#)

[TypeIntegerSpinWrap](#)

5. If you selected a block of cells, set the BlockMode property back to False.

[Print](#)

[Copy](#)

[Close](#)

### To create an owner-drawn cell

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. Set the [CellType](#) property to 11 (Owner Drawn).
4. Specify the style of the owner-drawn cell with the [TypeOwnerDrawStyle](#) property.  
This style is called by the [DrawItem](#) event, which occurs when the cell needs to be drawn.
5. If you selected a block of cells, set the [BlockMode](#) property back to False.

[Print](#)

[Copy](#)

[Close](#)

### To create a PIC cell

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. Set the [CellType](#) property to 4 (PIC).
4. Use the following properties to customize the PIC cell:
  - [TypePicDefaultText](#)
  - [TypeHAlign](#)
  - [TypePicMask](#)
5. If you selected a block of cells, set the BlockMode property back to False.



[Print](#)

[Copy](#)

[Close](#)

### To create a picture cell

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. Set the [CellType](#) property to 9 (Picture).
4. Use the following properties to customize the picture cell:

[TypePictPicture](#)

[TypePictCenter](#)

[TypePictStretch](#)

[TypePictMaintainScale](#)

5. If you selected a block of cells, set the BlockMode property back to False.

[Print](#)

[Copy](#)

[Close](#)

#### To create a static text cell

**Note** You cannot scroll through text in a static text cell. You can resize the cell to fit the text. For more information, see [Resizing Cells](#).

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. Set the [CellType](#) property to 5 (Static Text).
4. Use the following properties to customize the static text cell:

[TypeHAlign](#)

[TypeTextAlignVert](#)

[TypeTextPrefix](#)

[TypeTextShadow](#)

[TypeTextShadowIn](#)

[TypeTextWordWrap](#)

**Note** The [TypeTextShadow](#) and [TypeTextShadowIn](#) properties cannot both be set to True at the same time.

5. If you selected a block of cells, set the [BlockMode](#) property back to False.

[Print](#)

[Copy](#)

[Close](#)

### To create a time cell

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. Set the [CellType](#) property to 6 (Time).
4. Use the following properties to customize the time cell:

[TypeSpin](#)

[TypeTime24Hour](#)

[TypeHAlign](#)

[TypeTimeMax](#)

[TypeTimeMin](#)

[TypeTimeSeconds](#)

[TypeTimeSeparator](#)

5. If you selected a block of cells, set the [BlockMode](#) property back to False.

## Customizing Cell Borders

You can specify whether a cell or group of cells has a border. A group of cells can be a row, a column, a range of cells, or an entire spreadsheet. If cells have a border, it can be displayed on the left, right, top, or bottom, or around all four sides of a cell or cell range.

A border can be the spreadsheet grid (if displayed) or it can be displayed as a solid, dashed, dotted, dash-dot, or dash-dot-dot line in regular and fine line weights. You can also specify that no border is displayed.

You can also specify a border color. For more information on customizing border colors, see [Setting the Border Color](#).

— [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To customize cell borders

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. If you do not want to display a border around the cell or cell range, set the [CellBorderStyle](#) property to 0 (None).
4. If you want to display a border around the cell or cell range, specify the border type with the [CellBorderStyle](#) property.
5. Specify the border style with the [CellBorderStyle](#) property.
6. Set the [Action](#) property to 16 (Set Border).
7. If you selected a block of cells, set the [BlockMode](#) property back to False.

## Locking Cells

You can lock cells to prevent editing or resizing. If a cell is locked, you can unlock it, or you can disable all locked cells.

You can also prevent the user from resizing rows or columns so that a row or column cannot be resized with a mouse. For more information, see [Preventing Resizing of Columns and Rows with the Mouse](#).

- [Locking Cells to Prevent Editing](#)
- [Unlocking Locked Cells](#)
- [Removing Protection from Locked Cells](#)

## Locking Cells to Prevent Editing

You can lock an individual cell or a range of cells to prevent editing. Cells that are locked are not actually locked until the spreadsheet is *protected*. If the spreadsheet is protected, locked cells are locked and the user cannot edit them.

You can unlock an individual cell or a range of cells to allow editing. You can also remove protection from all locked cells to allow editing. For information on unlocking cells, see [Unlocking Locked Cells](#). For more information on removing spreadsheet protection, see [Removing Protection from Locked Cells](#).

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To lock cells

OCX

1. On the [Op Mode](#) property page,
  - a. Select the Protect Locked Cells check box.
  - b. Choose the Apply button.
2. At run time,
  - a. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate.
  - b. If you are selecting a block of cells, set the [BlockMode](#) property to True.
  - c. Set the [Lock](#) property to True.
  - d. If you selected a block of cells, set the BlockMode property back to False.



[Print](#)

[Copy](#)

[Close](#)

### To lock cells

VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. Set the [Protect](#) property to True.
4. Set the [Lock](#) property to True.
5. If you selected a block of cells, set the BlockMode property back to False.

## Unlocking Locked Cells

You can unlock a cell or a range of cells that are locked against editing.

- [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To unlock cells**

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. Set the [Lock](#) property to False.
4. If you selected a block of cells, set the BlockMode property back to False.

## Removing Protection from Locked Cells

If cells are locked to prevent editing, you can edit locked cells by removing spreadsheet protection.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To remove protection from locked cells**

OCX

On the [Op Mode](#) property page,

1. Clear the Protect Locked Cells check box.
2. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To remove protection from locked cells**

VBX

Set the [Protect](#) property to False.

## Resizing Cells

You cannot scroll through text in a cell. However, you can resize a cell to fit the widest or tallest data in that cell. When you change the width and height of a cell, the column width and row height also change. You can base the column width and row height on either of the following:

- The width and height of the data in a particular cell
  - The widest data in a column and the tallest data in a row
- Note** If you base the column width and row height on the width and height of a particular cell and you later enter data that is wider or taller in another cell in the same row or column, the data will be truncated.
- [Resizing a Cell to Fit the Widest Data](#)
  - [Resizing a Cell to Fit the Tallest Data](#)

## Resizing a Cell to Fit the Widest Data

If you resize a cell to fit the widest data, the column width changes accordingly.

- [OCX, VBX Instructions](#)



[Print](#)

[Copy](#)

[Close](#)

**To resize a cell based on the widest data in that cell**

OCX, VBX

At run time,

1. Define the cell you want to resize with the [Col](#) and [Row](#) properties.
2. Get the minimum width of the specified cell that is necessary to display all the text within that cell with the [MaxTextCellWidth](#) property and assign it to a variable.
3. Set the [ColWidth](#) property to the variable from Step 2.

**To resize a cell based on the widest data in that column**

OCX, VBX

At run time,

1. Get the minimum width that is necessary to display the widest data in a specific column with the [MaxTextColWidth](#) property (specify the column using the *Index* parameter) and assign it to a variable.
2. Set the [ColWidth](#) property for that column equal to the variable from Step 1.

## Resizing a Cell to Fit the Tallest Data

If you resize a cell to fit the tallest data, the row height changes accordingly.

— [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To resize a cell based on the tallest data in that cell**

OCX, VBX

At run time,

1. Define the cell you want to resize with the [Col](#) and [Row](#) properties.
2. Get the minimum height of the specified cell that is necessary to display all the text within that cell with the [MaxTextCellHeight](#) property and assign it to a variable.
3. Set the [RowHeight](#) property to the variable from Step 2.

**To resize a cell based on the tallest data in that row**

OCX, VBX

At run time,

1. Get the minimum height that is necessary to display the tallest data in a specific row with the [MaxTextRowHeight](#) property (specify the row using the *Index* parameter) and assign it to a variable.
2. Set the [RowHeight](#) property for that row equal to the variable from Step 1.

## Allowing Cells to Overflow

You can specify that the contents of a cell can overflow into an adjacent cell if that cell is empty.

**Note** The [AllowCellOverflow](#) property only affects data entered by the user after the property is set. Text entered before this property is set does not overflow.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To allow the contents of a cell to overflow into an adjacent empty cell**

OCX

On the [General](#) property page under Allow,

1. Select the Cell Overflow check box.
2. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To allow the contents of a cell to overflow into an adjacent empty cell**

VBX

Set the [AllowCellOverflow](#) property to True.

## Allowing the User to Drag and Drop Cells

You can specify whether the user can select a cell or range of cells and drag and drop them to a new location in the same spreadsheet. When the mouse button is released and the cell block is dropped, the [DragDropBlock](#) event occurs.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To allow the user to drag and drop cells**

OCX

On the [General](#) property page under Allow,

1. Select the Drag and Drop check box.
2. Choose the Apply button.



[Print](#)

[Copy](#)

[Close](#)

**To allow the user to drag and drop cells**

VBX

Set the [AllowDragDrop](#) property to True.

## Selecting Cells

Many spreadsheet properties and functions perform actions on a cell or range of cells. Before using these properties and functions, you must often select or specify the cell or group of cells.

Set the [Col](#) and [Row](#) properties to identify the coordinates of a cell.

- [Relationship of Col and Row Properties](#)
- [Customizing Cell Selection](#)

## Relationship of Col and Row Properties

The following table shows how the [Col](#) and [Row](#) properties are used in relation to one another.

Col Property Value	Row Property Value	Action is performed on . . .
-1	-1	The entire spreadsheet
-1	0	The header row across the top of the screen (column header row)
-1	nonzero	The specified row
0	-1	The header column down the left side of the screen (row header column)
0	0	The gray cell in the upper-left corner of the spreadsheet
0	nonzero	The header cell at the specified row within the row headers
nonzero	-1	The specified column
nonzero	0	The header cell at the specified column within the column headers
nonzero	nonzero	The specified cell

Many spreadsheet properties and functions rely on the [Col](#) and [Row](#) properties to specify the column and row on which to perform the desired operation. If an operation is to be performed on an entire column or row, for example, when setting the [CellType](#) property, the application should specify the entire column or row instead of performing the operation on each cell within the column or row. This optimizes speed and memory.

## Customizing Cell Selection

You can let the user select only columns, rows, blocks, all cells, or any combination of these.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To customize cell selection**

OCX

On the [Op Mode](#) property page under Select Block Options,

1. Select the appropriate check boxes.
2. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To customize cell selection**

VBX

Set the [SelectBlockOptions](#) property.

## Working with a Block of Cells

You use the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties to define a block of cells in the spreadsheet. You use the [BlockMode](#) property to specify that modifications can be made to the designated block.

For more information on how the Col and Row properties are used in relation to one another, see [Relationship of Col and Row Properties](#).

- [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To work with a block of cells**

OCX, VBX

At run time,

1. Set the [Col](#) and [Col2](#) properties.
2. Set the [Row](#) and [Row2](#) properties.
3. Set the [BlockMode](#) property to True.
4. Set the properties you need to accomplish your task.
5. Set the BlockMode property back to False.



## Selecting Multiple Blocks of Cells

Sometimes you might want to select several blocks of cells that are not adjacent to one another and therefore cannot be selected as one group. You can select multiple blocks of cells and you can let the user select multiple discontinuous blocks.

For more information on how to perform an action on multiple blocks of cells, see the example for [Action](#) property setting 18.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To select multiple blocks of cells**

OCX

1. On the [General](#) property page under Allow,
  - a. Select the Multiple Block Selections check box.
  - b. Choose the Apply button.
2. At run time,
  - a. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties.
  - b. Set the [Action](#) property to 17 (Add MultiSel Blocks).
  - c. Repeat Steps 2a and 2b until all blocks are selected.

At design time,

- a. Select a block with the mouse.
- b. Press Ctrl and select additional blocks with the mouse.

[Print](#)

[Copy](#)

[Close](#)

### **To select multiple blocks of cells**

VBX

At run time,

1. Set the [AllowMultiBlocks](#) property to True.
2. Select a cell or a cell range with the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties.
3. Set the [Action](#) property to 17 (Add MultiSel Blocks).
4. Repeat Steps 2 and 3 until all blocks are selected.

At design time,

1. Select a block with the mouse.
2. Press Ctrl and select additional blocks with the mouse.

## Setting the Font

You can set the font characteristics of a cell or block of cells.

- [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To set the font

OCX,VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. Set the appropriate font properties ([FontName](#), [FontSize](#), [FontBold](#), [FontItalic](#), [FontStrikethru](#), and [FontUnderline](#)).
4. If you selected a block of cells, set the BlockMode property back to False.

## Using Edit Mode

When the user types data in a cell in the spreadsheet, edit mode is automatically on. When the user moves from the cell, edit mode is automatically off. If edit mode is on, the active cell is the cell being edited.

You can manually turn edit mode on and off. You can specify whether edit mode is to remain on when the user switches between cells. This is useful if you want to emulate a table of edit controls.

You can also specify whether text in the cell is replaced as the user types or whether the cursor is placed at the end of the existing text and new characters are inserted at that point.

The [EditMode](#) event, with the *Mode* parameter set to 1 (True), is sent to the application each time the user enters edit mode. The [EditMode](#) event, with the *Mode* parameter set to 0 (False), is sent to the application each time the user leaves edit mode. An application might want to know when the user leaves edit mode so that validation can be performed on the data.

### Notes

- Picture cells, static text cells, and cells that are protected and locked cannot be edited.
- When the [EditModePermanent](#) property is set to True, it overrides the [EditMode](#) property and the focus rectangle is hidden (see [Hiding the Focus Rectangle](#)).
- [Turning edit mode on and off manually](#)
- [Keeping edit mode on when switching between cells](#)
- [Specifying text replacement or addition during edit mode](#)

[Print](#)

[Copy](#)

[Close](#)

### To turn edit mode on and off manually

OCX, VBX

At run time,

1. If you want to turn edit mode off, set the [EditMode](#) property to False.
2. If you want to turn edit mode on, set the EditMode property to True.

[Print](#)

[Copy](#)

[Close](#)

**To keep edit mode on when switching between cells**

OCX

On the [Edit Mode](#) property page under Edit Mode,

1. Select the Permanent check box.
2. Choose the Apply button.

VBX

Set the [EditModePermanent](#) property to True.



[Print](#)

[Copy](#)

[Close](#)

**To specify text replacement or addition during edit mode**

OCX

On the [Edit Mode](#) property page under Edit Mode,

1. If you want the existing cell text to be replaced, select the Replace Existing Text check box.
2. If you want the new cell text to be appended to the existing text, clear the Replace Existing Text check box.
3. Choose the Apply button.

VBX

1. If you want the existing cell text to be replaced, set the [EditModeReplace](#) property to True.
2. If you want the new cell text to be appended to the existing text, set the EditModeReplace property to False.

## Navigating within Cells

You can identify or set the active cell in a spreadsheet. The active cell is the cell that currently contains the focus rectangle.

During edit mode, you can specify if and where the focus rectangle moves when the user presses the Enter key. You can also hide the focus rectangle in edit mode.

You can also use the Tab key to move the focus rectangle to the next cell in the spreadsheet.

For more information about navigating within the spreadsheet, see [Navigation](#).

- [Identifying the Active Cell](#)
- [Specifying the Active Cell](#)
- [Setting the Focus Movement](#)
- [Using the Tab Key to Move the Focus](#)
- [Preventing Active Cell Change on Focus](#)
- [Hiding the Focus Rectangle](#)

## Identifying the Active Cell

The active cell is the cell that currently contains the focus rectangle. The application can return the coordinates of the active cell.

- [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To identify the active cell**

OCX, VBX

To retrieve the column and row numbers of the active cell, at run time, use the [ActiveCol](#) and [ActiveRow](#) properties, respectively.

## Specifying the Active Cell

You can specify the cell to be the active cell in the spreadsheet.

- [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To specify the active cell**

OCX, VBX

At run time,

1. Specify the cell coordinates with the [Col](#) and [Row](#) properties.
2. Set the [Action](#) property to 0 (Activate Cell).

## Setting the Focus Movement

You can control where the focus rectangle moves (up, down, left, right, next cell, previous cell, or first column of the next row) when the user presses the Enter key when edit mode is on. You can also specify that no action occurs or that the focus rectangle stays at the same cell.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To specify the focus movement**

OCX

On the [Edit Mode](#) property page under Enter Key Action,

1. If you want to specify that no processing occurs when the user presses the Enter key, select the None option button.
2. If you want to specify that the focus rectangle stays at the current cell when the user presses the Enter key, select the Same option button.
3. If you want to specify that the focus rectangle moves to another cell when the user presses the Enter key, select the appropriate option button (Up, Down, Left, Right, Next, Previous, or Next Row).
4. Choose the Apply button.



[Print](#)

[Copy](#)

[Close](#)

### **To specify the focus movement**

VBX

1. If you want to specify that no processing occurs when the user presses the Enter key, set the [EditEnterAction](#) property to 0 (None).
2. If you want to specify that the focus rectangle stays at the current cell when the user presses the Enter key, set the EditEnterAction property to 7 (Same).
3. If you want to specify that the focus rectangle moves to another cell when the user presses the Enter key, set the EditEnterAction property to the appropriate setting.

## Using the Tab Key to Move the Focus

You can specify whether the Tab key moves the focus rectangle to the next cell in the spreadsheet or to the next control on the dialog.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To use the Tab key to move the focus**

OCX

On the [Edit Mode](#) property page,

1. If you want to move the focus to the next control on the dialog, clear the Process Tab Key check box.
2. If you want to move the focus to the next cell to the right, select the Process Tab Key check box.
3. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To use the Tab key to move the focus**

VBX

1. If you want to move the focus to the next control on the dialog, set the [ProcessTab](#) property to False.
2. If you want to move the focus to the next cell to the right, set the ProcessTab property to True.

## Preventing Active Cell Change on Focus

By default, the active cell is moved to the location of the mouse pointer when the user moves the focus to the spreadsheet with the mouse. You can prevent the active cell from changing when the spreadsheet receives the focus.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To prevent the active cell from changing**

OCX

On the [General](#) property page,

1. Clear the On Focus Set Cell to Cursor check box.
2. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To prevent the active cell from changing**

VBX

Set the [MoveActiveOnFocus](#) property to False.

## Hiding the Focus Rectangle

You can hide the focus rectangle during edit mode.

- [OCX Instructions](#)
- [VBX Instructions](#)



[Print](#)

[Copy](#)

[Close](#)

**To hide the focus rectangle**

OCX

On the [Edit Mode](#) property page under Edit Mode,

1. Select the Permanent check box.
2. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To hide the focus rectangle**

VBX

Set the [EditModePermanent](#) property to True.

## Spread Control Features

FarPoint Technologies' Spread control provides a flexible, powerful tool for creating spreadsheets and tables. Several unique features help you customize your interface to meet your needs.

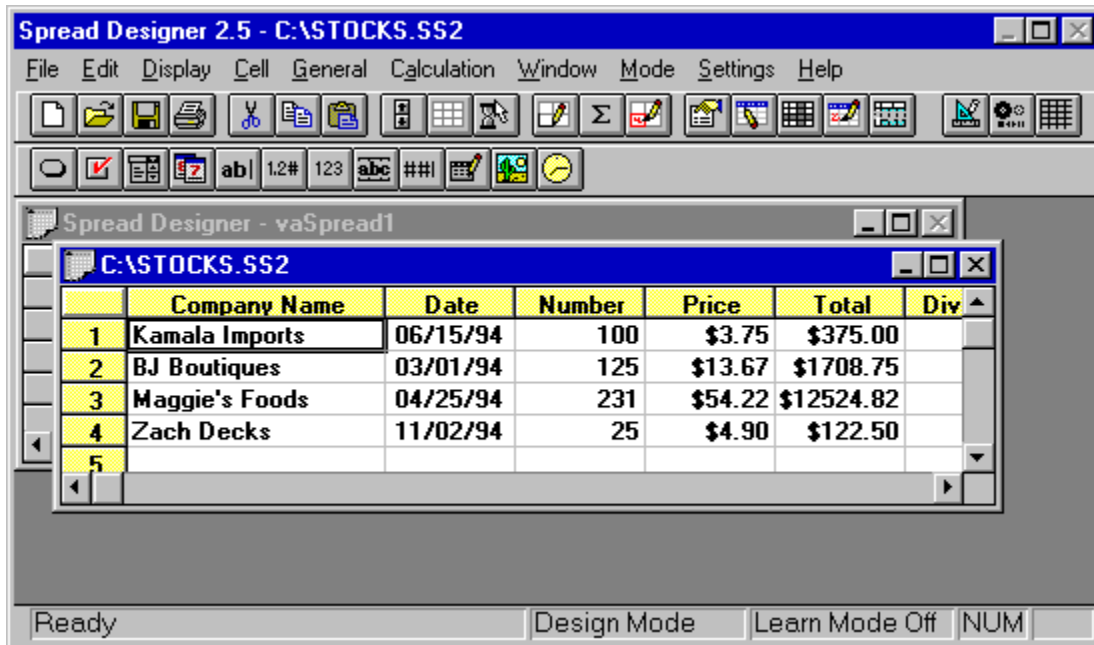
- [Spread Designer](#)
- [Printing](#)
- [Cells](#)
- [Columns and Rows](#)
- [Precedence in Property Settings](#)
- [Navigation](#)
- [Display](#)
- [Formulas](#)
- [Virtual Mode](#)
- [Data Binding](#)
- [Components of the OCX and VBX Controls](#)

## Spread Designer

The Spread Designer, formerly the Interface Designer, lets OCX and VBX users design spreadsheets quickly by offering a design-time environment that displays changes to the spreadsheet as they are made.

Within the Spread Designer, you can view a design-time version of your spreadsheet in design mode. You can also switch to a run-time version of the spreadsheet, which shows your spreadsheet as it will appear when it is in your application. For example, when working with hidden columns, you can view the columns and design them in design mode. When you switch to run-time mode, the columns are hidden.

The Spread Designer lets you open multiple spreadsheet views. The Spread Designer view is the default view provided by the designer. The Spread Designer view displays the spreadsheet in its current design-time state. You can open views of other existing spreadsheets created using FarPoint's Spread control, as shown in the following figure.



Once open, you can modify the Spread Designer view or one of the open views, and then choose which view, if any, to apply to the Spread control.

Using learn mode, you can generate documented Visual Basic code for actions you perform while working in the Spread Designer. You can view this code in the Spread Designer and you can paste it in your application.

Using cell type mode, you can see at a glance the type of cells in the spreadsheet.

For instructions for using the Spread Designer, see [Using the Spread Designer](#).

## Printing

Spread lets you customize your printed spreadsheet with a variety of options.

Your printed pages can include headers and footers. You can decide whether you want to print the spreadsheet border, the column header row and row header column, the spreadsheet grid, colors, and shadows in headers.

Spread lets you customize which portion of the spreadsheet is printed to your needs. You can print just the portion of the spreadsheet that contains data or specify a page range. You can place page breaks anywhere you need them.

Pages can print in either portrait or landscape orientation. In addition, you can specify left, right, top, and bottom margins for the printed pages. OCX and VBX users can also use the Smart Print option, which automatically adjusts the printing to accommodate the spreadsheet data.

For more information about printing spreadsheets, see [Printing a Spreadsheet](#).

## Cells

Cells can be any of 12 types: button, check box, combo box, date, edit, float, integer, owner-drawn, PIC, picture, static text, and time. You can tailor cells to display information as needed, and you can provide customized user interaction for any cell.

Each cell type presents special characteristics that you can customize for your needs. For example, you can specify maximum and minimum values, a separator character, and a format for date cells. The following figure illustrates date cells that have the format MM/DD/YY and that have a slash character as a separator.

Date
06/15/94
03/01/94
04/25/94
11/02/94

For more information about selecting a cell type for a cell, see [Designating the Cell Type](#).

Individual cells can display specialized borders and background and foreground colors. You can specify the alignment of text in a cell. In addition, cells can display text as overflowing into adjacent cells, if you prefer. For more information about customizing cells' appearance and cell text, see [Customizing Cell Borders](#) and [Allowing Cells to Overflow](#).

To customize user interaction, you can let the user drag and drop cells or blocks of cells. Depending on your needs, the user can select a cell, blocks of cells, only rows, one and only one row, or multiple rows. In addition, you can prevent the user from interacting with a cell by locking the cell. For more information about user interaction, see the following topics:

- [Locking Cells](#)
- [Resizing Cells](#)
- [Allowing the User to Drag and Drop Cells](#)
- [Selecting Cells](#)

Cells that allow user input can provide customized edit modes, such as automatically replacing existing text as the user types. For more information about edit mode, see [Using Edit Mode](#).

## Columns and Rows

Many of the custom features you can apply to an individual cell you can also specify for a column, a row, or a range of columns or rows. The spreadsheet also provides headers in the form of column header rows and row header columns that number the columns and rows or provide additional information about their content. In addition, you can freeze columns and rows, size them according to the data they contain, and hide them.

Header rows and columns can be customized as you need. Numbering can be displayed as numbers or letters, or you can display text in the header for a specific row or column. You can specify the background and foreground colors for the header rows and columns, and specify whether shadows are displayed and what colors are used in the shadows. If you prefer, you can hide header rows or columns.

The following figure illustrates a spreadsheet with customized column header rows and row header columns.

	<b>M</b>	<b>T</b>	<b>W</b>	<b>T</b>
<b>7:00</b>		<b>Meet the Teacher Day at Vicky's school</b>	<b>Vet. Appt. Tasha at 7:45</b>	
<b>8:00</b>	<b>Production meeting</b>	<b>Teacher - Parent conference at 8:15</b>		

For more information about customizing header rows and columns, see [Customizing the Column Header Row and Row Header Column](#).

Frozen columns do not scroll as the user scrolls through the spreadsheet. By default, the spreadsheet's column header row and row header column are frozen. You can specify additional adjacent columns and rows that do not scroll. For information about freezing columns and rows, see [Freezing Columns and Rows](#).

You can specify the width of a column and the height of a row, or you can let the dimensions vary depending on the data contained in the column or row. In addition, you can allow the user to resize columns and rows in your application. For information about working with column width and row height, see [Sizing Columns and Rows](#).

## **Precedence in Property Settings**

Cell settings override row, column, and spreadsheet settings. When Spread looks at a cell-related setting (for example, BackColor), it looks first at the cell setting. If Spread does not find a setting for the cell, it then looks at the row default setting. If not found, it then looks at the column default setting. If not found, it finally uses the spreadsheet default.



## Navigation

Spread provides multiple ways to navigate within the spreadsheet. Besides allowing you to set how the Tab and Enter keys are interpreted by your application, Spread supports the following navigation keys:

Key	Action
Up Arrow	Moves active cell up one row
Down Arrow	Moves active cell down one row
Right Arrow	Moves active cell right one column
Left Arrow	Moves active cell left one column
Shift+arrow key	Extends selection in direction of arrow key
PgUp	Moves active cell one page up
PgDn	Moves active cell one page down
Ctrl+PgUp	Moves active cell one page left
Ctrl+PgDn	Moves active cell one page right
Home	Moves active cell to first cell in row
End	Moves active cell to last cell in row that contains data
Ctrl+Home	Moves active cell to first row, first column
Ctrl+End	Moves active cell to last row and column that contain data
Tab	If the <a href="#">ProcessTab</a> property is set to True, moves active cell to next cell to the right (or down at end of row)
Shift+Tab	If the ProcessTab property is set to True, moves active cell to next cell to the left (or up at beginning of row)
Shift+Space	Selects current row
Ctrl+Space	Selects current column
Shift+Ctrl+Space	Selects entire spreadsheet
Shift+Del or Ctrl+X	Cuts current selection or active cell's data to Clipboard
Shift+Ins or Ctrl+V	Pastes Clipboard contents into active cell
Ctrl+Ins or Ctrl+C	Copies current selection or active cell's data to Clipboard
Enter	Depends on setting of the <a href="#">EditEnterAction</a> property
Esc	If spreadsheet is in edit mode, previous cell value replaces new value and edit mode is turned off
F2	If edit mode is on, cell value is cleared
F3	If edit mode is on in a date or time cell, current date or time is placed in cell
F4	If edit mode is on in a date cell, spreadsheet displays a pop-up calendar to let you choose a date

## Display

You can customize the spreadsheet display to create colorful spreadsheets, tables, and more. If you prefer, your spreadsheet need not display an outer border. You can also choose whether to display horizontal or vertical grid lines or both. If grid lines are displayed, you can specify their color and style. A spreadsheet with customized grid lines is shown in the following figure.

1st Year	2nd Year	3rd Year	4th Year
English I	English II	Brit. Lit.	Amer. Lit.
History	Economics	Polit. Sci.	Amer. Hist.
Spanish I	Spanish II	Spanish III	Spanish IV
Free	Chemistry	Free	Physics
Biology	Free	Biology II	Calculus
Algebra	Geometry	Algebra II	Chorus
Chorus	Art	Drama	Dance
Phys. Ed.	Phys. Ed.	Dance	Free

Scroll bars can be displayed all the time or only when needed. You may want to display a special vertical scroll bar that displays scroll arrows rather than a scroll box, particularly if you are using virtual mode. For more information about scroll bars, see [Displaying Scroll Bars](#) and [Displaying the Special Vertical Scroll Bar](#).

You can ensure that the spreadsheet only displays complete columns or rows rather than partial ones. In addition, you can specify how the spreadsheet aligns with the outer borders of the control and how many columns or rows are displayed at one time.

You can customize many additional display features, including the color of the area behind the spreadsheet in the Spread control, how the user selects items in the spreadsheet and how selected items are displayed, and where buttons appear in the

spreadsheet.

For information and instructions on customizing spreadsheet display, see [Customizing the Spreadsheet Display](#).

## Formulas

With Spread you can provide formulas in cells, taking advantage of Spread's Calc Engine. In addition, Spread offers relative and absolute cell referencing. You can use the operations provided by the Calc Engine, or you can create your own functions to use in formulas.

For more information about using formulas, see [Entering Formulas](#).

## Virtual Mode

Virtual mode lets the Spread control read in only the amount of data it needs to display the requested rows. Therefore, virtual mode increases responsiveness and conserves system resources.

When the Spread control is in virtual mode and the user scrolls through spreadsheet rows, rows are read and stored in a buffer until they are displayed. Spread lets you customize virtual mode to fit your needs: you can specify how many rows are retrieved into the buffer and how many displayed rows are retained in the buffer.

For more information about virtual mode, see [Using Virtual Mode](#).

## Data Binding

You can bind the Spread control to Visual Basic's data control to display database records. Many spreadsheet characteristics can be set automatically when the database is read by the Spread control, including column header text, column size, and cell type. Alternatively, you can tailor these and other characteristics to meet your needs, including optimizing performance by using virtual mode.

The following figure illustrates a Spread control bound to a database. The Spread control has automatically configured itself to display the database records.

	<b>Author</b>	<b>Title</b>	<b>Year</b>	<b>ISBN</b>
1	Anne Ball	Alex: A parrot's tale	1995	0-0280875-2-
2	Lydia Kernville	Curly tails and white paws	1994	0-0480095-2-
3	Julia Shell	Clyde's adventures	1994	0-0275195-2-
4	Russell Lumky	Dog training for children	1995	0-0283095-4-
5	Robby Dixie-Smith	Weimaraners: Your friends	1992	0-3256095-2-
6	Scott Peterson	Caddy: A boy and his dog	1992	0-0453495-4-
7	Chris Persons	Iguanas, the perfect pets	1993	0-3256325-2-
8	Edith Anne Leach	The care and feeding of puppies	1996	0-3258695-2-

For more information about data binding, see [Binding the Spreadsheet to a Database](#).

## Components of the OCX and VBX Controls

If you are using the Spread OCX or VBX control, properties and events let you customize the control for your interface. Functions and methods provide VBX and OCX users the opportunity to set multiple characteristics at one time.

If you are using the Spread OCX control, access the property pages for the control to customize it. Property pages provide a familiar notebook interface that organizes properties by task. For example, all the properties for customizing edit mode are available on one page. For descriptions of each property page available for OCX users, see [Spread Control Property Pages](#).

Note that if you are using the OCX, you can access some properties on property pages, but others may not be available or may be available only through the property list, depending on your container.

- [Properties](#)
- [Events](#)
- [Functions and Methods](#)

## Using the Spread Designer

- [Overview](#)
- [Starting the Spread Designer](#)
- [Design Mode Versus Run-Time Mode](#)
- [Using Learn Mode](#)
- [Using Cell Type Mode](#)
- [Opening a Spreadsheet](#)
- [Spread Designer View Versus Active View](#)
- [Saving and Applying Changes](#)
- [Setting Up the Spread Designer Environment](#)
- [Specifying the Cell Type in the Spread Designer](#)
- [Data Binding in the Spread Designer](#)

## Overview: Spread Designer

You can use the Spread Designer at design time to design the look of the Spread control. You can set both design-time and run-time properties. By setting properties at design time instead of run time, you can view the run-time appearance of the spreadsheet and your application will run faster because less code is being executed during the Form\_Load event.

The Spread Designer creates a "snapshot" of the Spread control. Once all changes are made, you apply those changes to the selected Spread control on your form or dialog.

**Note** You can run the Spread Designer as a stand-alone program to create spreadsheet files. However, you cannot set data-binding properties when running the Spread Designer as a stand-alone program.



## Starting the Spread Designer

Before you use the Spread Designer, you should be familiar with the Spread features described in [Spread Control Features](#).

- [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To start the Spread Designer

1. If you are using the Spread OCX control, display the pop-up menu by clicking the Spread control with the right mouse button and then choose Spread Designer.
2. If you are using the Spread VBX control,
  - a. Place a Spread control on the form or dialog.
  - b. Double-click the InterfaceDesigner property in the Properties window or dialog box.
3. If you want to run the Spread Designer as a stand-alone program, type:

C:\...\SD32D25.EXE

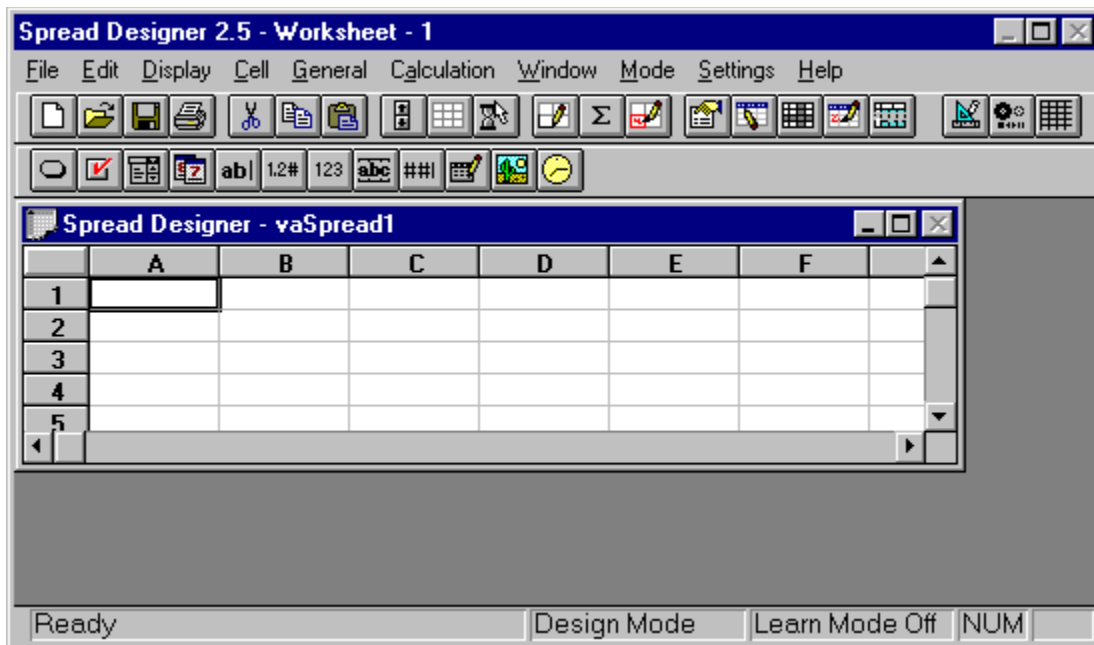
if you are using a 32-bit development environment, or type:

C:\...\SD16D25.EXE

if you are using a 16-bit development environment.

The preceding instructions assume that the drive is C and that you supply the path where the Spread Designer executable file resides.

The following window appears:



You can quickly access most of the drop-down menus by using the toolbar buttons. The following table describes the toolbar buttons:

scroll bars	grid lines	pointers	border lines	formulas	cell colors
spreadsheet environment	headers	columns and rows	spreadsheet colors	virtual mode	run-time mode
design mode (on)	learn mode (off)	learn mode (on)	cell type mode (off)	cell type mode (on)	button
check box	combo box	date	edit	float	integer
owner-drawn	PIC (mask)	picture	static text (label)	time	

## Design Mode Versus Run-Time Mode

In the Spread Designer you can be in either *design mode* or *run-time mode*. By default, when you start the Spread Designer you are operating in design mode.

In design mode, the "design" spreadsheet displays in a WYSIWYG format where certain properties are *temporarily* reset to values that optimize the spreadsheet design process. The following table shows the properties that are temporarily reset while in design mode.

Property	Setting while in design mode
<a href="#">AllowDragDrop</a>	True
<a href="#">AllowMultiBlocks</a>	True
<a href="#">AllowUserFormulas</a>	True
<a href="#">AutoClipboard</a>	True
<a href="#">DisplayColHeaders</a>	True
<a href="#">DisplayRowHeaders</a>	True
<a href="#">Lock</a>	True
<a href="#">MoveActiveOnFocus</a>	True
<a href="#">OperationMode</a>	SS_OP_MODE_NORMAL
<a href="#">Protect</a>	False
<a href="#">RetainSelBlock</a>	True
<a href="#">ScrollBars</a>	SS_SCROLLBAR_BOTH
<a href="#">SelectBlockOptions</a>	SS_SELBLOCKOPT_BLOCKS   SS_SELBLOCKOPT_COLS   SS_SELBLOCKOPT_ROWS   SS_SELBLOCKOPT_ALL
<a href="#">UserResize</a>	SS_USER_RESIZE_COL   SS_USER_RESIZE_ROW
<a href="#">UserResizeCol</a>	SS_USER_RESIZE_DEFAULT
<a href="#">UserResizeRow</a>	SS_USER_RESIZE_DEFAULT
<a href="#">VisibleCols</a>	0
<a href="#">VisibleRows</a>	0

For example, if you hide a row, it will still display in the Spread Designer and you can still work with that row.

In run-time mode, the "design" spreadsheet displays in a WYSIWYG format where all properties you have set using the Spread Designer, both design-time and run-time, are applied as they are currently set. For example, if you hide a row, it will not display in the Spread Designer and you cannot work with that row.

You can toggle between design and run-time modes. When you switch to run-time mode, the spreadsheet reflects the original property settings. Design mode does not permanently change the property values you have set.

— [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To switch between design mode and run-time mode**

Do one of the following:

- From the Mode menu, choose Design Mode or Run-Time Mode.
- Click the run-time button on the feature toolbar.  
This toggle button switches the Spread Designer between design mode and run-time mode.

## Using Learn Mode

You can record the actions you perform in the Spread Designer with *learn mode*. Learn mode generates Visual Basic code and writes that code to the Clipboard. You can view the code on the Clipboard and you can paste the code in your project.

You can use learn mode to generate code for an unfamiliar property or to test a new application.

**Tip** The information that is recorded during learn mode is stored in an internal buffer while learn mode is active. When you exit learn mode, the buffer immediately places the information on the system Clipboard. You should move the information on the Clipboard to your application or another file as soon as possible to avoid accidental deletion.

— [OCX, VBX Instructions](#)

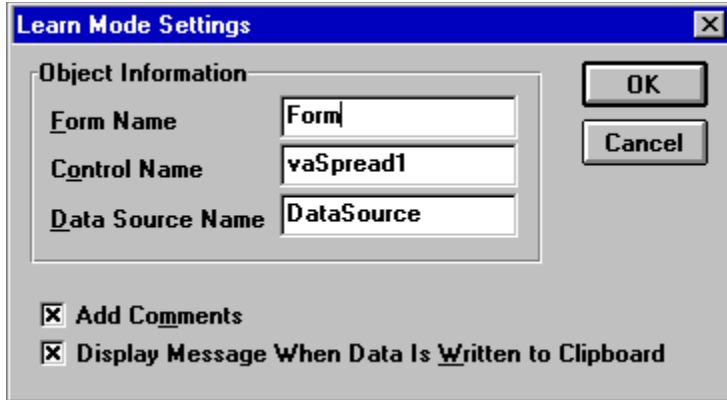
[Print](#)

[Copy](#)

[Close](#)

#### To use learn mode

1. Start the Spread Designer.
2. From the Settings menu, choose Learn Mode.  
The Learn Mode Settings dialog box appears.



**Learn Mode Settings**

**Object Information**

**Form Name**

**Control Name**

**Data Source Name**

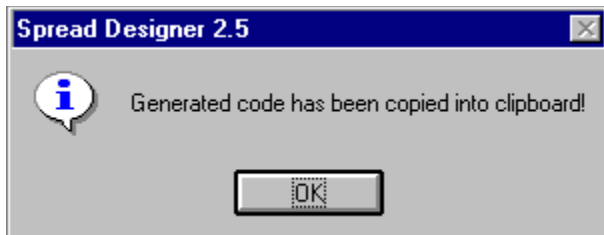
**Add Comments**

**Display Message When Data Is Written to Clipboard**

**OK**

**Cancel**

3. Under Object Information,
  - a. Type the name of the form containing the Spread control in the Form Name box.
  - b. Type the name of the Spread control in the Control Name box.
  - c. If the Spread control is bound to a database, type the name of the data control in the Data Source Name box.
4. If you do not want comments in the generated code, clear the Add Comments check box.
5. If you do not want a message displayed to confirm when the code is written to the Clipboard, clear the Display Message When Data Is Written to Clipboard check box.
6. Choose the OK button.
7. To start learn mode, do one of the following:
  - From the Mode menu, choose Learn Mode On.
  - Click the learn mode button on the toolbar.This toggle button switches learn mode on and off.
8. Perform actions using the Spread Designer.
9. From the Mode menu, choose Learn Mode Off or click the learn mode button on the toolbar.  
If you chose to display a confirmation message in Step 5, it appears.



10. Choose the OK button.
11. To view the generated code, from the Window menu, choose Clipboard Viewer.
12. To paste the code from the Clipboard in your application, do the following:
  - a. Click an insertion point in your application where you want to paste the code.
  - b. Press Shift+Insert to paste the code.

## Using Cell Type Mode

In *cell type mode*, the date, edit, float, integer, owner-drawn, PIC (mask), picture, static text (label), and time cell types are each indicated by a different background color. (No colors are displayed for the button, check box, and combo box cell types because their type is self-evident.) This allows you to see at a glance what type a cell is.

Cell background colors are temporarily overridden while the Spread Designer is in cell type mode.

**Note** When the Spread Designer is in cell type mode, you cannot save a spreadsheet, apply the changes back to the original, selected control, or set cell colors. If you attempt to perform any of these actions, you will be prompted to exit cell type mode.

— [OCX, VBX Instructions](#)

[Print](#)

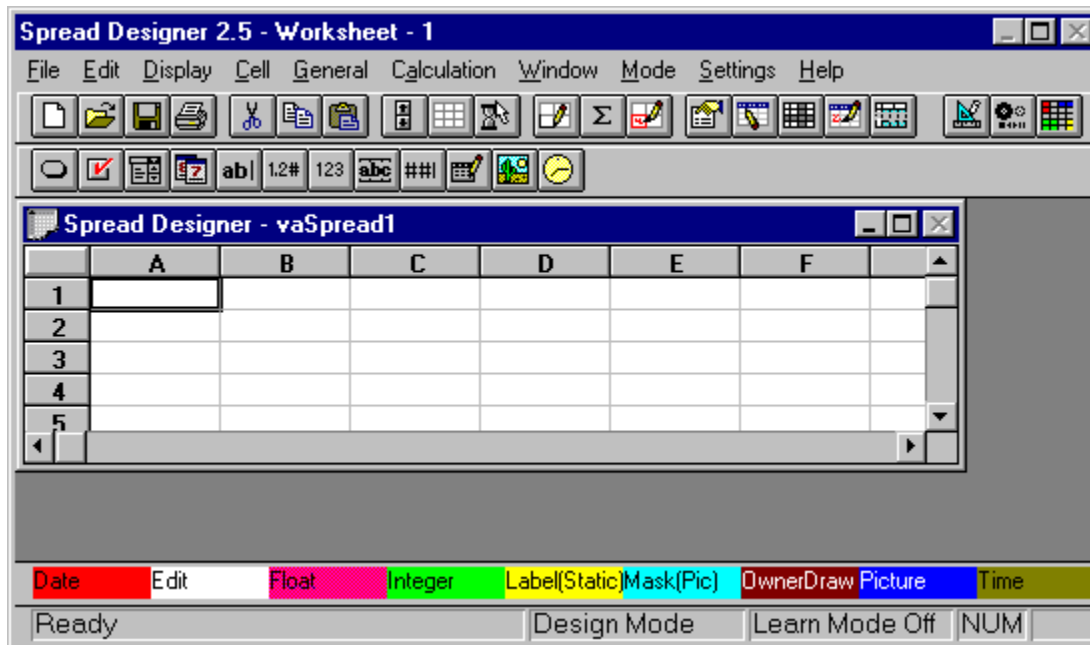
[Copy](#)

[Close](#)

### To use cell type mode

1. Start the Spread Designer.
2. To start cell type mode, do one of the following:
  - From the Mode menu, choose Cell Type Mode On.
  - Click the cell type mode button on the toolbar.
    - This toggle button switches cell type mode on and off.

The cell type indication bar appears above the status bar and previously set date, edit, float, integer, owner-drawn, PIC (mask), picture, static text (label), and time cell types are displayed with the appropriate background colors.



3. When you are ready to exit cell type mode, do one of the following:
  - From the Mode menu, choose Cell Type Mode Off.
  - Click the cell type mode button on the toolbar.



## Opening a Spreadsheet in the Spread Designer

You can open more than one spreadsheet at a time in the Spread Designer. By default, spreadsheets created using the Spread Designer have an extension of .SS2 (binary) or .TB2 (tab-delimited). However, you can use any extension.

**Note** Keep in mind the following if you are using data binding:

- If you start the Spread Designer from the command line, the Spread Designer does not recognize any spreadsheet as being data bound.
- The Spread Designer only recognizes the Spread Designer view as being data bound (see [Spread Designer View Versus Active View](#)).
- [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To open a spreadsheet**

1. From the File menu, choose Open.
2. In the Open dialog box, specify the path and filename and choose the OK button.

The spreadsheet appears in the Spread Designer window.

## Spread Designer View Versus Active View

When you select a Spread control and start the Spread Designer, the spreadsheet view displayed is the *Spread Designer view*. The title bar of the Spread Designer view reads "Spread Designer - *control name*". The Spread Designer creates a "snapshot" of the selected Spread control.

If you have more than one spreadsheet open in the Spread Designer, the spreadsheet that has the focus is called the *active view*. Therefore, the Spread Designer view can also be the active view. Conversely, the active view does not have to be the Spread Designer view.

You can apply the changes from either the Spread Designer view or the active view to the selected Spread control (on the form). For more information on applying changes made in the Spread Designer, see [Customizing How Changes Are Applied](#).

To set data-binding properties, the Spread Designer view must be the active view and the Spread control must be linked to a data source control on the form. You cannot set data-binding properties for a spreadsheet that you open in the Spread Designer.

## Saving and Applying Changes

When you are satisfied with your spreadsheet design, you can

- Save your changes to a spreadsheet file that you can open in the Spread Designer and use as a template or to a spreadsheet file that you can load.
- Apply your changes back to the Spread control and continue working in the Spread Designer.
- Apply your changes and exit the Spread Designer.

**Caution** If you do not save or apply your changes, all changes will be lost.

### Notes

- If you have more than one Spread control on your form or dialog, the changes are only applied back to the original, selected Spread control.
- The changes you apply back to the original, selected Spread control are not permanent until you save the Form file.
- If you exit the Spread Designer without choosing Save or Apply first, you might or might not be prompted to apply your changes, depending on how you have set up your Spread Designer environment. For more information, see [Customizing How Changes Are Applied](#).
- [Saving changes to a spreadsheet file](#)
- [Applying changes to the selected control](#)

[Print](#)

[Copy](#)

[Close](#)

**To save changes to a spreadsheet file**

1. If you want to save format only, from the File menu, choose Save As, and select .TB2 as the extension.
2. If you want to save format and data, from the File menu, choose Save As, and select .SS2 as the extension.
3. If you want to save format and data and use a different extension, type the filename in the Filename box.

[Print](#)

[Copy](#)

[Close](#)

**To apply changes to the selected control**

1. If you want to apply the changes before you exit the Spread Designer,
  - a. To save format and data, from the File menu, choose Apply, All (Both Data and Format).
  - b. To save format only, from the File menu, choose Apply, Format.
  - c. To save data only, from the File menu, choose Apply, Data.
2. If you want to apply changes when you exit the Spread Designer, from the File menu, choose Apply & Exit or choose Exit Spread Designer.

**Caution** If you choose not to be prompted to apply your changes and you exit the Spread Designer without saving or applying your changes, all changes will be lost.

## Setting Up the Spread Designer Environment

You can customize how changes you make are applied back to the selected Spread control. You can also customize the Spread Designer balloon help.

- [Customizing How Changes Are Applied](#)
- [Customizing Balloon Help](#)

## Customizing How Changes Are Applied

When applying changes, you can specify whether to apply the Spread Designer view or the active view to the selected Spread control. If you choose to apply the active view, you can do so with or without verification.

You can apply the changes to the selected control before you exit the Spread Designer or you can apply them when you exit. For more information, see [Saving and Applying Changes](#).

If you exit the Spread Designer without choosing Save or Apply first, you can specify whether a dialog box appears that prompts you to apply your changes. When you exit the Spread Designer without doing a Save or Apply and the dialog box appears prompting you to apply your changes, you can specify whether format, data, or both are applied.

**Caution** If you do not display a dialog box and you exit the Spread Designer without saving or applying your changes, all changes will be lost.

— [OCX, VBX Instructions](#)



[Print](#)

[Copy](#)

[Close](#)

**To customize how changes are applied**

1. If you want to specify the view applied to the selected control,
  - a. From the Settings menu, choose General.
  - b. Select an option button under View Applied to Control.
2. If you want to display a dialog box when exiting the Spread Designer without saving or applying changes,
  - a. From the Settings menu, choose General.
  - b. Select the Verify When Closing Window without Saving/Applying check box.
3. If you want to specify what is applied at exit verification,
  - a. From the Settings menu, choose General.
  - b. Select an option button under Type Applied by Exit Verification.

## Customizing Balloon Help

Balloon help is available for the feature and cell type toolbar buttons. You can set whether this help displays. If the help displays, you can customize the shape, tail type, and shadow characteristics of the balloon.

- [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To customize balloon help**

1. From the Settings menu, choose General.
2. To hide the balloon help, clear the Display Balloons check box under Balloon Settings.
3. To change the balloon shape, select an option button under Balloon Settings, Shape.
4. To change the balloon tail, select an option button under Balloon Settings, Tail Type.
5. To change the shadow behind the balloon, select an option button under Balloon Settings, Shadow.

## Specifying the Cell Type in the Spread Designer

Cell types define the type of information that appears in a cell. You can specify the cell type for rows, columns, individual cells, or selected groups of cells in a spreadsheet. For a summary of cell types, see [Designating the Cell Type](#).

- [Creating a button cell](#)
- [Creating a check box cell](#)
- [Creating a combo box cell](#)
- [Creating a date cell](#)
- [Creating an edit cell](#)
- [Creating a float cell](#)
- [Creating an integer cell](#)
- [Creating an owner-drawn cell](#)
- [Creating a PIC cell](#)
- [Creating a picture cell](#)
- [Creating a static text cell](#)
- [Creating a time cell](#)

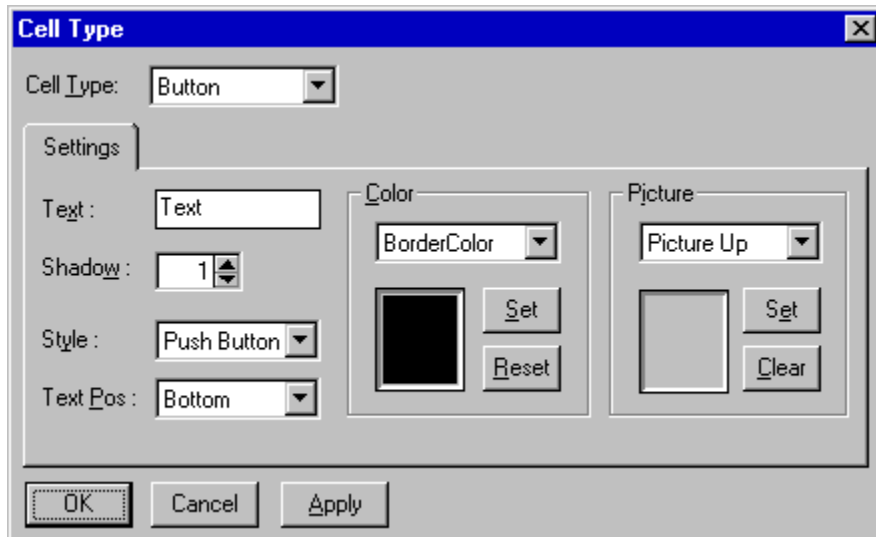
[Print](#)

[Copy](#)

[Close](#)

### To create a button cell

1. Select a cell or block of cells.
2. Choose the cell type by using one of the following methods:
  - Display the pop-up menu by right-clicking the cell or block of cells and then choose Button.
  - From the Cell menu, choose Cell Type and choose Button.
  - Choose the button button on the cell type toolbar.The Cell Type dialog box appears.



3. Select the text and graphics alignment from the Text Pos drop-down list box.
4. To specify button colors, under Color,
  - a. Select one of the following from the drop-down list box:
    - BorderColor (border)
    - ButtonColor (background)
    - DarkColor (dark shadow)
    - LightColor (light shadow)
    - TextColor (text)
  - b. Choose the Set button.
  - c. Specify the color by selecting one of the displayed colors or define a custom color.
  - d. Choose the OK button.
5. Type the button shadow size or click the arrows in the Shadow spin box.
6. To specify button pictures, under Picture,
  - a. Select one of the following from the drop-down list box:
    - Picture Up (displays when button is in up position)
    - Picture Down (displays when button is in down position)
  - b. Choose the Set button to display the file selection dialog box.
  - c. Select an available graphics file.
  - d. Choose the Open button.
7. Type the button text in the Text box.
8. Select the type of button from the Style drop-down list box.
9. Choose the Apply button.

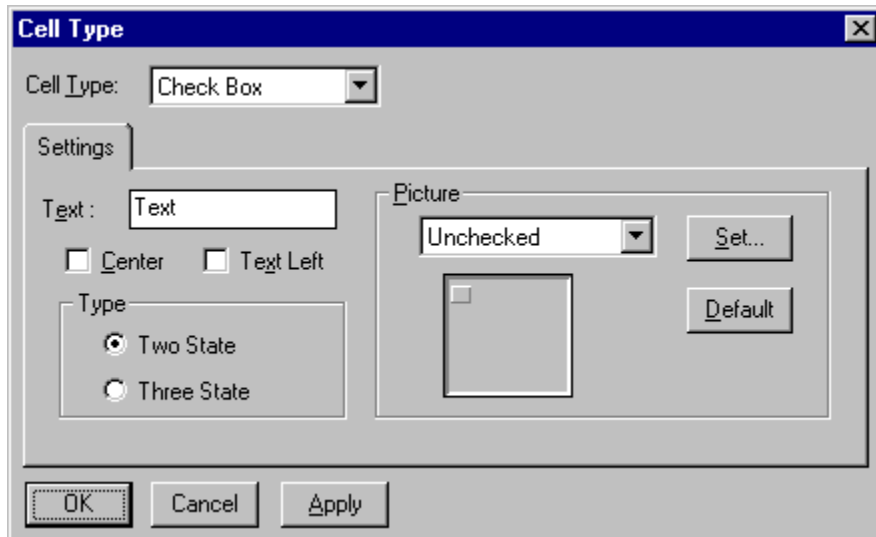
[Print](#)

[Copy](#)

[Close](#)

### To create a check box cell

1. Select a cell or block of cells.
  2. Choose the cell type by using one of the following methods:
    - Display the pop-up menu by right-clicking the cell or block of cells and then choose Checkbox.
    - From the Cell menu, choose Cell Type and choose Checkbox.
    - Choose the check box button on the cell type toolbar.
- The Cell Type dialog box appears.



3. Type the check box text in the Text box.
4. To center the check box in the cell, select the Center check box.
5. To align the text to the left of the pictures, select the Text Left check box.
6. To specify a three-state check box, under Type, select the Three State option button.
7. To specify pictures for different states of the check box, under Picture,
  - a. Select one of the following from the drop-down list box:
    - Unchecked  
—displays when check box is cleared (False state)
    - Unchecked Down  
—displays when check box is cleared (False state) and pressed
    - Checked  
—displays when check box is selected (True state)
    - Checked Down  
—displays when check box is selected (True state) and pressed
    - Grayed  
—displays when check box is grayed (Indeterminate state)
    - Grayed Down  
—displays when check box is grayed (Indeterminate state) and pressed
  - b. Choose the Set button to display the file selection dialog box.
  - c. Select an available graphics file.
  - d. Choose the Open button.
8. Choose the Apply button.

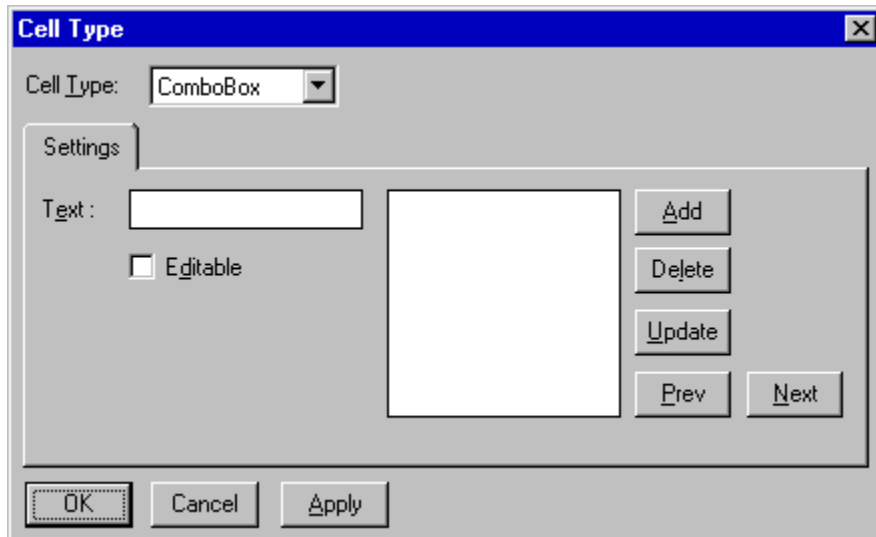
[Print](#)

[Copy](#)

[Close](#)

### To create a combo box cell

1. Select a cell or block of cells.
  2. Choose the cell type by using one of the following methods:
    - Display the pop-up menu by right-clicking the cell or block of cells and then choose Combobox.
    - From the Cell menu, choose Cell Type and choose Combobox.
    - Choose the combo box button on the cell type toolbar.
- The Cell Type dialog box appears.



3. To set whether the combo box text is editable in the spreadsheet, select the Editable check box.
4. Type the combo box list items in the Text box.
5. Choose the Add button.
6. To delete a combo box list item in the Cell Type dialog box,
  - a. Click the item or choose the Prev or Next buttons to highlight the item.
  - b. Choose the Delete button.
7. To edit a combo box list item in the Cell Type dialog box,
  - a. Click the item to select it.
  - b. Type the changes in the Text box.
  - c. Choose the Update button.
8. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

### To create a date cell

1. Select a cell or block of cells.
  2. Choose the cell type by using one of the following methods:
    - Display the pop-up menu by right-clicking the cell or block of cells and then choose Date.
    - From the Cell menu, choose Cell Type and choose Date.
    - Choose the date button on the cell type toolbar.
- The Cell Type dialog box appears.

Cell Type

Cell Type: Date

Settings

Format: YY/MM/DD

Min Date: 01/01/1868

Max Date: 12/31/2099

Separator: /

Text Align

Left  Show Century

Center  Spin Button

Right

OK Cancel Apply

3. To display the year in century format, select the Show Century check box.
4. Select the date format from the Format drop-down list box.
5. Type the maximum allowable date in the Max Date box.
6. Type the minimum allowable date in the Min Date box.
7. Type the character that separates the day, month, and year in the Separator box.
8. To display a spin button, select the Spin Button check box.
9. To specify text alignment, select an option button under Text Align.
10. Choose the Apply button.



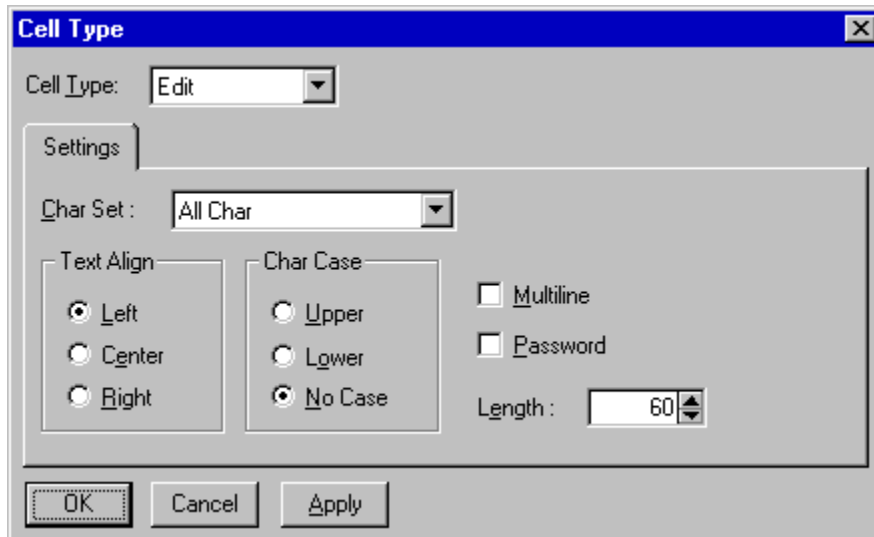
[Print](#)

[Copy](#)

[Close](#)

#### To create an edit cell

1. Select a cell or block of cells.
  2. Choose the cell type by using one of the following methods:
    - Display the pop-up menu by right-clicking the cell or block of cells and then choose Edit.
    - From the Cell menu, choose Cell Type and choose Edit.
    - Choose the edit button on the cell type toolbar.
- The Cell Type dialog box appears.



3. Specify the case of the characters by selecting an option button under Char Case.
4. Select a character set that can be used to type data into the cell from the Char Set drop-down list box.
5. Click the arrows or type a value in the Length spin box to specify the maximum number of characters allowed in the edit field.
6. Specify that multiple lines of data can be typed in the cell by selecting the Multiline check box.
7. Specify that the user can set up a password in the cell (the edit field displays the '\*' character instead of the actual typed character) by selecting the Password check box.
8. Specify text alignment by selecting an option button under Text Align.
9. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

### To create a float cell

1. Select a cell or block of cells.
  2. Choose the cell type by using one of the following methods:
    - Display the pop-up menu by right-clicking the cell or block of cells and then choose Float.
    - From the Cell menu, choose Cell Type and choose Float.
    - Choose the float button on the cell type toolbar.
- The Cell Type dialog box appears.

**Cell Type**

Cell Type:

**Settings**

Min Value :

Max Value :

Int Digits :

Dec Digits :

**Text Align**

Left

Center

Right

Money

Separator

Decimal

3. Display a currency type by selecting the Money check box and typing the character to display in the Money box.
4. Display a thousands separator character (to separate the thousands place from the hundreds place) by selecting the Separator check box and typing the character to display in the Separator box.
5. Specify the number of digits that can be typed to the right of the decimal place by clicking the arrows or typing the value in the Dec Digits spin box.
6. Specify the number of digits that can be typed to the left of the decimal place by clicking the arrows or typing the value in the Int Digits spin box.
7. Type the maximum allowable value in the Max Value box.
8. Type the minimum allowable value in the Min Value box.
9. Specify data alignment by selecting an option button under Text Align.
10. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

### To create an integer cell

1. Select a cell or block of cells.
  2. Choose the cell type by using one of the following methods:
    - Display the pop-up menu by right-clicking the cell or block of cells and then choose Integer.
    - From the Cell menu, choose Cell Type and choose Integer.
    - Choose the integer button on the cell type toolbar.
- The Cell Type dialog box appears.

Cell Type

Cell Type: Integer

Settings

Min Value : -99999999

Max Value : 99999999

Text Align

Left

Center

Right

Spin Settings

Use Spin Button

Inc Count : 1

Spin Wrap

OK Cancel Apply

3. Type the maximum allowable value in the Max Value box.
4. Type the minimum allowable value in the Min Value box.
5. Display a spin button in the cell by selecting the Use Spin Button check box.
6. If a spin button is displayed, specify the amount by which the value in the cell is incremented or decremented when the spin button is clicked by clicking the arrows or typing the value in the Inc Count spin box.
7. If a spin button is displayed, specify that the value in the cell should wrap when the minimum or maximum values are reached by selecting the Spin Wrap check box.
8. Specify data alignment by selecting an option button under Text Align.
9. Choose the Apply button.

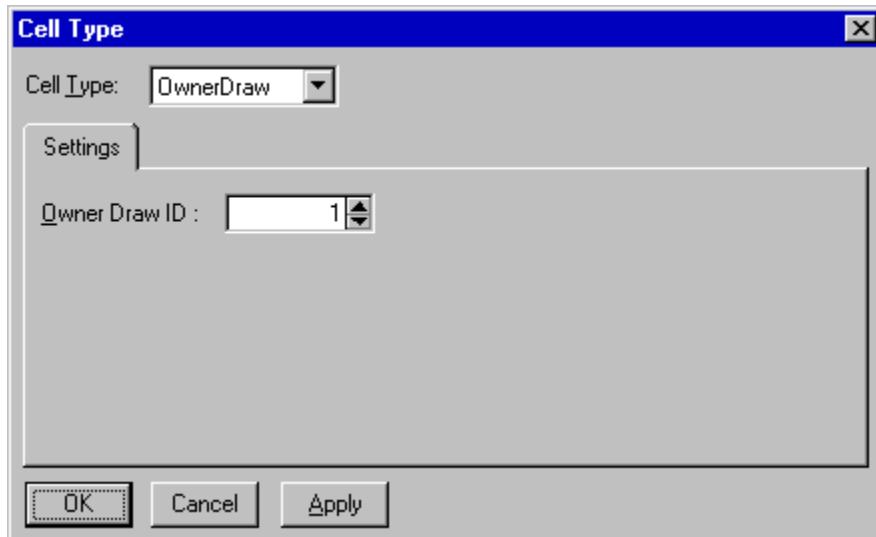
[Print](#)

[Copy](#)

[Close](#)

### To create an owner-drawn cell

1. Select a cell or block of cells.
  2. Choose the cell type by using one of the following methods:
    - Display the pop-up menu by right-clicking the cell or block of cells and then choose Owner Draw.
    - From the Cell menu, choose Cell Type and choose Owner Draw.
    - Choose the owner-drawn button on the cell type toolbar.
- The Cell Type dialog box appears.



3. Click the arrows or type a value in the Owner Draw ID spin box to specify an identification number for an owner-drawn cell.
4. Choose the Apply button.

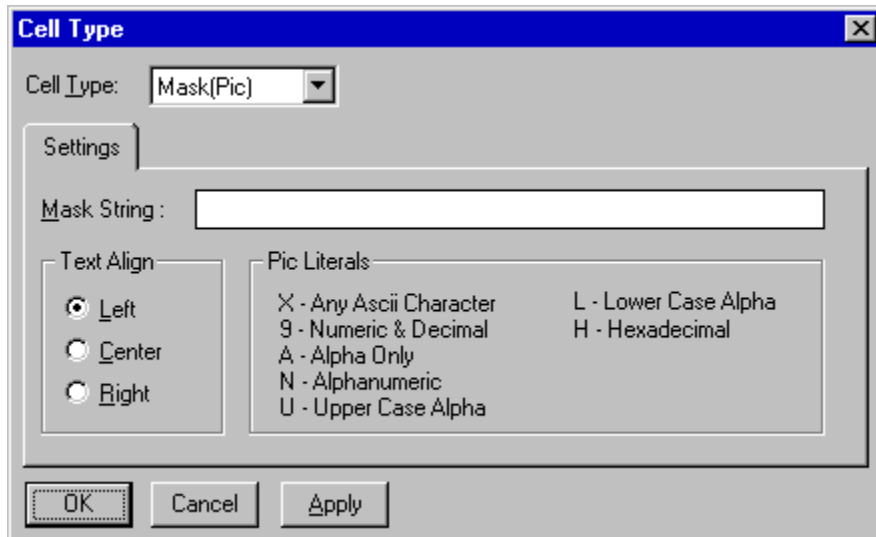
[Print](#)

[Copy](#)

[Close](#)

#### To create a PIC cell

1. Select a cell or block of cells.
  2. Choose the cell type by using one of the following methods:
    - Display the pop-up menu by right-clicking the cell or block of cells and then choose Mask(Pic).
    - From the Cell menu, choose Cell Type and choose Mask(Pic).
    - Choose the PIC (Mask) button on the cell type toolbar.
- The Cell Type dialog box appears.



3. Type the mask used with the PIC formatted edit field in the Mask String box.
4. Specify text alignment by selecting an option button under Text Align.
5. Choose the Apply button.

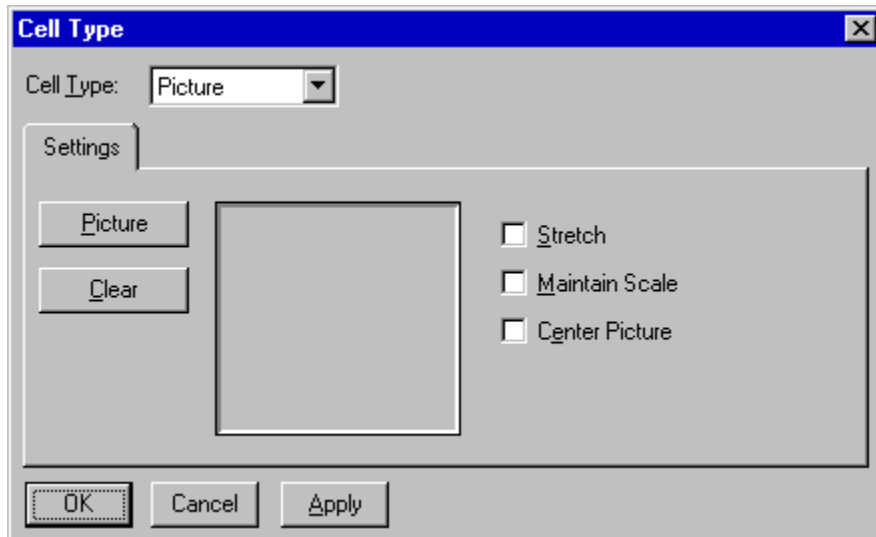
[Print](#)

[Copy](#)

[Close](#)

### To create a picture cell

1. Select a cell or block of cells.
  2. Choose the cell type by using one of the following methods:
    - Display the pop-up menu by right-clicking the cell or block of cells and then choose Picture.
    - From the Cell menu, choose Cell Type and choose Picture.
    - Choose the picture button on the cell type toolbar.
- The Cell Type dialog box appears.



3. Choose the Picture button to display the file selection dialog box.
4. Select an available graphics file.
5. Center the picture in the cell by selecting the Center Picture check box.
6. Stretch the picture to fit the size of the cell by selecting the Stretch check box.
7. If the picture is stretched, maintain the proportionality by selecting the Maintain Scale check box.
8. Choose the Apply button.

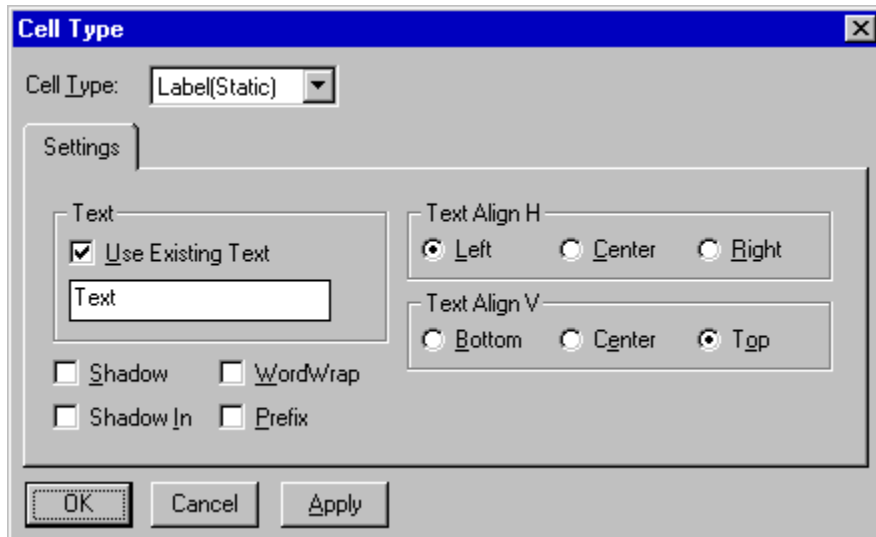
[Print](#)

[Copy](#)

[Close](#)

#### To create a static text cell

1. Select a cell or block of cells.
  2. Choose the cell type by using one of the following methods:
    - Display the pop-up menu by right-clicking the cell or block of cells and then choose Label(Static).
    - From the Cell menu, choose Cell Type and choose Label(Static).
    - Choose the static text (label) button on the cell type toolbar.
- The Cell Type dialog box appears.



3. Specify the horizontal text alignment by selecting an option button under Text Align H.
4. Specify the vertical text alignment by selecting an option button under Text Align V.
5. Type the text in the Text box.
6. If you want to underline a character in the text, select the Prefix check box and type an ampersand (&) in front of the character you want to underline.
7. Display a shadow effect inside the cell by selecting the Shadow check box.
8. Display an inverted shadow effect inside the cell by selecting the Shadow In check box.
9. Specify that text wraps in the cell if the text string is wider than the cell width by selecting the WordWrap check box.
10. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To create a time cell**

1. Select a cell or block of cells.
2. Choose the cell type by using one of the following methods:
  - Display the pop-up menu by right-clicking the cell or block of cells and then choose Time.
  - From the Cell menu, choose Cell Type and choose Time.
  - Choose the time button on the cell type toolbar.The Cell Type dialog box appears.
- 
3. To display a spin button in the cell, select the Spin Button check box.
4. Select the 12- or 24-hour time format from the Format drop-down list box.
5. Type the maximum allowable time value in the Max Time box.
6. Type the minimum allowable time value in the Min Time box.
7. Display seconds by selecting the Show Seconds check box.
8. Type the character used to separate the hours, minutes, and seconds in the Separator box.
9. Specify the text alignment by selecting an option button under Text Align.
10. Choose the Apply button.



## Data Binding in the Spread Designer

### Notes

- You can bind OCX and VBX Spread controls in Visual Basic only.
- The Spread control must be connected to a data source control.
- To set data-binding properties, the Spread Designer view must be the active view. You cannot set data-binding properties for a spreadsheet that you open in the Spread Designer (that is, by choosing Open from the File menu).
- If you bind columns to data fields using the Spread Designer and then change the record source for the data control, you must refresh the data control (for example, run your project) before resetting data fields with the Spread Designer. Unless the data control is refreshed, the new data fields will not be reflected in the Spread Designer.

Once you connect the Spread control to a database, you can customize the following data-binding features:

- Bind selected fields instead of the entire database.
- Supply custom text for headers.
- Size columns automatically or manually.
- Set the cell types for bound spreadsheets automatically or manually.
- Synchronize the active row of the spreadsheet and data control.
- Specify whether records modified in the spreadsheet are written to the database.
- Specify whether the Spread control automatically reads the data and displays each record in the spreadsheet.

For information on how to connect the Spread control to a database, see [Connecting to a Database Using Default Settings](#).

- [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To customize data-binding features

1. From the General menu, choose Data Binding.

The following window appears:

The screenshot shows the 'Data Binding' dialog box. It features a title bar with a close button. The main area is divided into two sections. The top section, titled 'Selected Column Data Binding Information', contains a 'Database Column (DataField)' dropdown menu, 'Set Data Field' and 'Clear Data Field' buttons, and a 'Fire Data Fill Event for Column' checkbox. Below this is a spreadsheet grid with columns A through G and rows 1 through 4. The bottom section contains several checkboxes: 'Set Cell Type Automatically', 'Automatically Fill Data', 'Inform Data Control on Active Row Changes', 'Automatically Save Data', 'Use DataField Name As Column Heading', and 'Size Column To Data'.

2. To bind specific fields,
  - a. Click a column to select it.
  - b. Under Selected Column Data Binding Information, select a database field from the Database Column drop-down list box.
  - c. Choose the Set Data Field button.
  - d. If you want to activate the [DataFill](#) event for each cell in the column when the data is read and when the data is written back to the database, select the Fire Data Fill Event for Column check box.
3. To customize header text,
  - a. Clear the Use DataField Name As Column Heading check box.
  - b. Specify the header text using the procedure outlined in [Specifying Header Text](#).
4. To size columns manually,
  - a. Clear the Size Column To Data check box.
  - b. At run time, specify the column width with the [ColWidth](#) property.
5. To manually set the cell types for bound spreadsheets,
  - a. Clear the Set Cell Type Automatically check box.
  - b. In the active view, select a cell or block of cells.
  - c. Set the cell type by using one of the following methods:
    - Right-click the cell or block of cells to display a pop-up menu and select the cell type.
    - From the Cell menu, choose Cell Type and choose the cell type.
    - Choose the appropriate button on the cell type toolbar.
6. To move through records in the spreadsheet independently of records in the database, clear the Inform Data Control on Active Row Changes check box.
7. To manually save modified spreadsheet records to the database,
  - a. Clear the Automatically Save Data check box.
  - b. At run time, set the [Action](#) property to 15 (Data Save).

8. To manually enter data from the database into the spreadsheet,
  - a. Clear the Automatically Fill Data check box.
  - b. At run time, load data in the spreadsheet.

## Binding the Spreadsheet to a Database

- [Connecting to a Database Using Default Settings](#)
- [Displaying Selected Fields in Spreadsheet Columns](#)
- [Customizing Headers for Bound Spreadsheets](#)
- [Sizing Columns in Bound Spreadsheets](#)
- [Assigning Cell Types for Bound Spreadsheets](#)
- [Specifying Spreadsheet and Database Interaction](#)
- [Using Virtual Mode with Bound Spreadsheets](#)

## Connecting to a Database Using Default Settings

Using Visual Basic, you can bind OCX and VBX Spread controls to databases. You can quickly bind the spreadsheet to the entire database, or you can customize data display and control interaction. You can also customize data binding in the Spread Designer. For more information, see [Data Binding in the Spread Designer](#).

When you bind a spreadsheet to a database using the default settings, the fields in the database table are assigned to spreadsheet columns, and the field names are displayed as the column headers.

**Note** If you are binding to a RemoteData control, the interface between the controls is read-only. You cannot write changes back to the database bound through the RemoteData control. Future versions of Spread might support write capabilities. For more information about the RemoteData control, refer to the Visual Basic documentation.

— [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To bind a spreadsheet to a database using default settings**

OCX, VBX

1. Create a data control on the form.
2. Set the DatabaseName property for the data control.
3. Set the RecordSource property for the data control.
4. Create a Spread control on the form.
5. Set the [DataSource](#) property to the name of the data control.

## Displaying Selected Fields in Spreadsheet Columns

The spreadsheet can display all the fields in the database, or it can display selected fields in designated columns.

- [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### **To display a database field in a specific column**

OCX, VBX

1. Follow the steps listed for [Connecting to a Database Using Default Settings](#) to bind the spreadsheet to a database.
2. At run time, set the [Col](#) property to designate the first column to display a specific field.
3. At run time, set the [DataField](#) property to the field name in the database table.
4. Repeat Steps 2 and 3 for each column for which you want to specify a field.

To see the spreadsheet display the bound field, use Visual Basic's Refresh method.



## Sizing Columns in Bound Spreadsheets

By default, columns are automatically sized based on the length of the largest string in the corresponding bound database field. You can manually size columns, or you can have them automatically sized based on the size of the bound database fields.

- [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To size columns in bound spreadsheets

OCX, VBX

1. Follow the steps listed for [Connecting to a Database Using Default Settings](#) to bind the spreadsheet to a database.
  2. To have the columns automatically sized based on the data in the database fields, set the [DAutoSizeCols](#) property to either of the following values:
    - 1 (Max Col Width) to size the column to the length of the longest string in the field
    - 2 (Best Guess) to size the column to the length of the database field
- Note** If the bound cells are edit cells, be aware that the default setting of the [TypeEditLen](#) property is 60 characters. You must reset the TypeEditLen property to 255 characters to accommodate a text field in the bound database.
3. If you do not want the columns automatically sized based on the data in the database fields, in the [DataColConfig](#) event, set the [DAutoSizeCols](#) property to 0 (Off) and if you want, size the columns manually.

For instructions for sizing columns, see [Sizing Columns and Rows](#).

## Customizing Headers for Bound Spreadsheets

By default, column headers display database field names. However, you can change the column headers to display your own text.

**Note** Once you have created a customized header for one column, the other columns will no longer display the database field names. Instead, they will display letters or numbers, as determined by the [ColHeaderDisplay](#) property.

— [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To display customized column headers

OCX, VBX

1. Follow the steps listed for [Connecting to a Database Using Default Settings](#) to bind the spreadsheet to a database.
2. Set the [DAutoHeadings](#) property to False.
3. At run time, specify the column header in which to display the custom text by setting the [Col](#) and [Row](#) properties.
4. At run time, supply the text for the column header using the [Text](#) property.

## Assigning Cell Types for Bound Spreadsheets

By default, the spreadsheet automatically assigns the cell type for each column's cells based on the field type of the corresponding bound database field. You can override the assigned cell type, or you can turn off the automatic assignment and assign the cell types manually.

- [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To assign cell types to a bound spreadsheet

OCX, VBX

1. Follow the steps listed for [Connecting to a Database Using Default Settings](#) to bind the spreadsheet to a database.
2. If you want to automatically assign the cell type to each column, set the [DAutoCellTypes](#) property to True.  
After each column is assigned a cell type, the [DataColConfig](#) event occurs.
3. If you want to override the automatically assigned cell type, within the DataColConfig event,
  - a. Optionally, you can return the assigned cell type using the [CellType](#) property at run time.
  - b. At run time, change the assigned cell type by specifying the column with the [Col](#) property, setting the [Row](#) property to —1, and then setting the CellType property to the desired type.

## Specifying Spreadsheet and Database Interaction

You can specify whether or not the spreadsheet and database synchronize the active record. You can move through records in the spreadsheet independently of records in the database, or the spreadsheet and database can move through records synchronously.

When the user makes changes to a spreadsheet record, you can specify whether the changes are automatically written to the bound database table.

— [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To specify spreadsheet and database interaction

OCX, VBX

1. Follow the steps listed for [Connecting to a Database Using Default Settings](#) to bind the spreadsheet to a database.
2. To synchronize the active row in the spreadsheet and database, set the [DInformActiveRowChange](#) property to True.
3. To automatically save modified spreadsheet records to the database, set the [DAutoSave](#) property to True.
4. To manually save modified spreadsheet records to the database,
  - a. Set the DAutoSave property to False.
  - b. At run time, set the [Action](#) property to 15 (Data Save).



## Using Virtual Mode with Bound Spreadsheets

Detailed instructions for using virtual mode are provided in [Using Virtual Mode](#). However, if you are using virtual mode with a bound spreadsheet, you should be aware that the following interactions might not behave as you would expect:

- Querying the number of records  
If you ask the data control how many records are in the database, it will tell you the number of records in the virtual buffer, not the number of records in the database.
- Searching the database  
If you search for data in the database, the data control will not look in every record. It will only look in records that are in the virtual buffer.

As a rule, using virtual mode with a bound Spread control limits the data control to working with only the information in the virtual buffer, not the entire database. Keep this in mind when designing interactions with the data control.

## Optimizing Your Spreadsheet

- [Overview](#)
- [Entering Formulas](#)
- [Disabling the Clipboard Shortcut Keys](#)
- [Specifying the Operation Mode](#)
- [Monitoring Data Changes in the Spreadsheet](#)
- [Using Virtual Mode](#)

## Overview: Optimizing Your Spreadsheet

You can use formulas in spreadsheet cells, and you can let users add their own formulas to cells. In addition, you can optimize the responsiveness of the spreadsheet and spreadsheet operations by

- Using and customizing spreadsheet formulas
- Disabling the Clipboard shortcut keys
- Specifying the operation mode
- Monitoring data changes in the spreadsheet
- Using virtual mode

For additional information about optimizing your spreadsheet, including techniques to save time and conserve system resources, you might want to see the following topics:

[Adding Data to a Spreadsheet](#)

[Copying Data](#)

[Clearing Data](#)

[Relationship of Col and Row Properties](#)

[Preventing the Spreadsheet from Flickering](#)

## Entering Formulas

You can define formulas (mathematical expressions) for cells in a spreadsheet. The formulas can use cell references to calculate values depending on the contents of the specified cells. You can specify how the cell references should be represented and how they should operate.

The Spread control has not only 9 formula operators (such as division and logical Or), but also 13 functions (such as absolute value and summation). For more information on the operators and functions available, see the [Formula](#) property.

- [Specifying Cell References](#)
- [Adding Formulas to a Spreadsheet](#)
- [Creating and Using Custom Formulas](#)
- [Allowing or Preventing User-Entered Formulas](#)
- [Automatically Recalculating and Updating Formulas](#)
- [Using Circular References in Formulas](#)

## Specifying Cell References

The Spread control can use *absolute* or *relative* cell references. In absolute cell references, for example, "B2", the column and row specified are the column and row used for calculations. In relative cell references, the column and row used for calculations change based on the location of the formula. For example, if you copy a formula using relative cell references from column B to column C, cell references such as "B2" change to "C2" to reflect the new formula location.

Note that if you use the "A1" style of cell references (similar to Microsoft Excel), the cell in which you place the formula is the cell used for relative addressing purposes.

- [QCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To specify the style of cell references**

OCX

Call the [SetRefStyle](#) method.

[Print](#)

[Copy](#)

[Close](#)

**To specify the style of cell references**

VBX

Call the [SpreadSetRefStyle](#) function.

## Adding Formulas to a Spreadsheet

You can add a formula to any cell in a spreadsheet. You can also let the user enter formulas directly into the spreadsheet. For more information on letting the user add formulas, see [Allowing or Preventing User-Entered Formulas](#).

- [OCX, VBX Instructions](#)



[Print](#)

[Copy](#)

[Close](#)

### To add a formula to a spreadsheet

OCX, VBX

At run time,

1. Specify the cell or cells to which you want to add a formula with the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. Define the formula with the [Formula](#) property.
4. If you selected a block of cells, set the BlockMode property back to False.

## Creating and Using Custom Formulas

To create and use custom formulas in your spreadsheet, you must define a custom function. For example, Spread does not provide a cube root operation. If you would like to provide that operation, you can define a custom function to do so.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To create and use a custom function

OCX

1. If you want to define a custom function with a specified number of parameters, call the [AddCustomFunction](#) method.
2. If you want to define a custom function with a variable number of parameters, call the [AddCustomFunctionExt](#) method.
3. Define the response to the [CustomFunction](#) event.

[Print](#)

[Copy](#)

[Close](#)

### To create and use a custom function

VBX

1. If you want to define a custom function with a specified number of parameters, call the [SpreadAddCustomFunction](#) function.
2. If you want to define a custom function with a variable number of parameters, call the [SpreadAddCustomFunctionExt](#) function.
3. Define the response to the [CustomFunction](#) event.

## Allowing or Preventing User-Entered Formulas

You can specify whether the user can type formulas into float or integer cells.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To allow or prevent user-entered formulas**

OCX

On the [General](#) property page under Allow,

1. Select the Users to Enter Formulas check box.
2. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

### To allow or prevent user-entered formulas

VBX

1. If you want to let the user enter formulas, set the [AllowUserFormulas](#) property to True.
2. If you want to prevent the user from entering formulas, set the AllowUserFormulas property to False.

## Automatically Recalculating and Updating Formulas

By default, the spreadsheet recalculates formulas in the spreadsheet when the contents of dependent cells change. You can turn this recalculation off. You can turn off automatic formula calculation.

By default, the spreadsheet updates formulas that use relative cell references when you insert or delete columns or rows or when you move or swap blocks of cells. You can turn off automatic formula updates.

If your formulas use absolute cell references, you must update formulas manually.

- [OCX Instructions](#)
- [VBX Instructions](#)



[Print](#)

[Copy](#)

[Close](#)

**To turn off automatic formula calculation and updates**

OCX

1. If you want to turn off automatic formula calculation, on the [General](#) property page under Calculations,
  - a. Clear the Automatic Calculations check box.
  - b. Choose the Apply button.
2. If you want to turn off automatic formula updates, on the [General](#) property page under Calculations,
  - a. Clear the Synchronize Formulas check box.
  - b. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

### To turn off automatic formula calculation and updates

VBX

1. If you want to turn off automatic formula calculation, set the [AutoCalc](#) property to False.
2. If you want to turn off automatic formula updates, call the [SpreadSetFormulaSync](#) function, and set the value of the *Sync* parameter to False.

## Using Circular References in Formulas

You can specify how the spreadsheet calculates circular references used in formulas. A circular reference occurs when a formula uses the value in a cell to compute another value, which in turn is used by the first cell to recalculate its value. A simple example is if cell A1's value is equal to the formula B1+C1, and the value of cell C1 is equal to A1\*3.

You can control the number of times the spreadsheet calculates a circular reference and whether it stops calculations after the value changes by less than the given amount.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To customize circular references**

OCX

Call the [SetIteration](#) method.

[Print](#)

[Copy](#)

[Close](#)

**To customize circular references**

VBX

Call the [SpreadSetIteration](#) function.

## Disabling the Clipboard Shortcut Keys

You can use Clipboard shortcut keys to cut (Ctrl+X), copy (Ctrl+C), and paste (Ctrl+V) data in the spreadsheet. You can disable the Clipboard shortcut keys.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To disable the Clipboard shortcut keys**

OCX

On the [General](#) property page,

1. Clear the Handle Clipboard Keys check box.
2. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To disable the Clipboard shortcut keys**

VBX

Set the [AutoClipboard](#) property to False.



## Specifying the Operation Mode

Operation modes control user interaction with the spreadsheet. The operation mode you apply limits user interaction such as moving the focus and selecting cells. Choose from the following operation modes:

<b>Mode</b>	<b>Description</b>
Normal	The default spreadsheet mode.
Read only	Cells, rows, and columns cannot receive the focus.
Row	Clicking a row moves the focus to that row. Double-clicking a cell moves the focus to that cell.
Single-selection	Clicking a row selects the row. One and only one row is selected at all times. The user cannot move the focus to an individual cell.
Multiple-selection	Clicking a row selects the row. The user can click additional rows to select them. Clicking a selected row deselects it.
Extended-selection	Clicking a row selects the row. The user can click while pressing Ctrl to select additional rows. The user can click and press Shift to select a range of rows. Pressing Ctrl and clicking a selected row deselects it.

**Note** For best results when using a spreadsheet in virtual mode, do not use multiple- or extended-selection operation mode.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To specify an operation mode**

OCX

On the [Op Mode](#) property page under Operation Mode,

1. Choose the operation mode by selecting the appropriate option button.
2. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To specify an operation mode**

VBX

Specify the operation mode using the [OperationMode](#) property.

## Monitoring Data Changes in the Spreadsheet

You can specify that any time data changes in a cell, a column, a row, or the entire spreadsheet, the change is noted. You might want to do this for bound spreadsheets to check for and write changed data to the bound database.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To monitor data changes

OCX

1. Using the [SetCellDirtyFlag](#) method when the spreadsheet is bound,
  - a. Set the value of the *Col* parameter to the column number of the cell to monitor, or set the value to `-1` to specify an entire row.
  - b. Set the value of the *Row* parameter to the row number of the cell or cells to monitor.
  - c. Set the value of the *Dirty* parameter to `True` to flag changed cells.
2. Use the [GetCellDirtyFlag](#) method to check whether a specific cell's data has been changed.

[Print](#)

[Copy](#)

[Close](#)

### To monitor data changes

VBX

1. Using the [SpreadSetCellDirtyFlag](#) function,
  - a. Set the value of the *Col* parameter to the column number of the cell to monitor, or set the value to  $-1$  to specify an entire row.
  - b. Set the value of the *Row* parameter to the row number of the cell or cells to monitor.
  - c. Set the value of the *Dirty* parameter to True to flag changed cells.
2. Use the [SpreadGetCellDirtyFlag](#) function to check whether a specific cell's data has been changed.

## Using Virtual Mode

Use virtual mode to increase Spread control responsiveness when working with large amounts of data. You can use virtual mode for a spreadsheet with a large number of rows or for a spreadsheet bound to a large database.

If you do not use virtual mode, the spreadsheet can take a long time to display data as you scroll through rows. The diminished responsiveness occurs because the spreadsheet must read through all the rows in the spreadsheet before displaying additional rows. In virtual mode, the spreadsheet reads ahead only when needed to display additional rows and retrieves records from a buffer area for faster response.

You can specify characteristics of virtual mode, such as how many rows are read into the buffer area at a time. In addition, you can display a special vertical scroll bar to use with virtual mode.

### Notes

Keep in mind the following when using virtual mode:

- If you are using a bound spreadsheet that is using virtual mode, the Data control bound to the spreadsheet might not behave as you expect. For more information, see [Using Virtual Mode with Bound Spreadsheets](#).
- When printing a spreadsheet that is using virtual mode, set the [PrintUseDataMax](#) property to False.
- Do not use the [Action](#) property setting 32 (Smart Print) to print a spreadsheet that is using virtual mode. Use the Action property setting 13 (Print) instead.
- If the spreadsheet does not know the total number of rows, the row header column might not display the correct numbers for the rows. For best results, hide the row header column.
- For best results, do not use multiple- or extended-selection operation mode in a spreadsheet that is using virtual mode.
- Do not freeze rows in a spreadsheet that is using virtual mode.
- You cannot sort a spreadsheet that is using virtual mode.
- [Specifying the Number of Rows](#)
- [Customizing Virtual Mode](#)
- [Displaying the Special Vertical Scroll Bar](#)

## Specifying the Number of Rows

When using virtual mode, the maximum number of rows in the spreadsheet, set using the [MaxRows](#) property, is no longer valid. You can specify the maximum number of rows for virtual mode using the [VirtualMaxRows](#) property.

If you add rows to a spreadsheet that is using virtual mode after all the rows have been read, the [VirtualMaxRows](#) property will be set to the original number of rows and the Spread control will not be able to read all the rows in the spreadsheet. You must set the value of the [VirtualMaxRows](#) property to `-1` to have it reset the number of rows, or you must provide the new number of rows.

If you are using a bound Spread control, once the database has been read and a value assigned to the [VirtualMaxRows](#) property, that value will not change until either you assign a new value to the [VirtualMaxRows](#) property or a new database is bound to the Spread control and read. If you bind a database after first binding and reading a smaller database, the [VirtualMaxRows](#) property will be set to the number of records in the smaller database and the Spread control will not be able to read all the records in the larger database.

— [OCX Instructions](#)

— [VBX Instructions](#)



[Print](#)

[Copy](#)

[Close](#)

**To specify the number of rows when using virtual mode**

OCX

On the [Virtual Mode](#) property page,

1. Specify the number of rows in the Virtual Max Rows box as follows:
  - If you know the exact number of rows in the spreadsheet or bound database, specify that number.
  - If you do not know the exact number of rows in the spreadsheet or bound database, but would like the spreadsheet to determine it once it has read all the rows, specify
    - 1.
  - If you do not know the exact number of rows in the spreadsheet or bound database, but would like to estimate it, overestimate the number rather than underestimating it.
    - If you underestimate the number, the spreadsheet never reads the records past the value you assign.
2. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To specify the number of rows when using virtual mode**

VBX

Set the [VirtualMaxRows](#) property as follows:

- If you know the exact number of rows in the spreadsheet or bound database, specify that number.
- If you do not know the exact number of rows in the spreadsheet or bound database, but would like the spreadsheet to determine it once it has read all the rows, specify  
—1.
- If you do not know the exact number of rows in the spreadsheet or bound database, but would like to estimate it, overestimate the number rather than underestimating it.  
If you underestimate the number, the spreadsheet never reads the records past the value you assign.

## Customizing Virtual Mode

You can specify how many rows are read into the virtual buffer at a time and whether some of the previously displayed rows are kept in the buffer when additional rows are read.

In addition, you can customize the vertical scroll bar to work more smoothly with virtual mode. For information about displaying a special vertical scroll bar, see [Displaying the Special Vertical Scroll Bar](#).

If the spreadsheet does not know the number of rows, the row header column might not display the correct numbers. For best results, hide the row header column. To hide the row header column, see [Hiding the Column Header Row and the Row Header Column](#).

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

## To customize virtual mode

OCX

On the [Virtual Mode](#) property page,

1. Select the Virtual Mode check box.
2. Specify the number of rows read into the virtual buffer at a time in the Rows Requested Each Time box.  
If the spreadsheet can display more rows at a time than the number of rows specified, the value in the Rows Requested Each Time box changes to the number of rows the spreadsheet can display.
3. Specify the total number of previously retrieved rows that are retained in a buffer in the Viewed Records to Keep in Memory box.
4. Under Vertical Scroll Bar, select the Box Reflects Rows in Buffer check box to have the vertical scroll box reflect only the number of rows in the virtual buffer rather than the total number of rows.
5. If you do not select the Box Reflects Rows in Buffer check box or if you are displaying the row header column, specify a number in the Virtual Max Rows box to provide the spreadsheet with the number of rows in the spreadsheet or bound database.  
Providing the number of rows helps the vertical scroll bar more accurately reflect the current position in the spreadsheet and permits the row header column to accurately number the rows. For more information about specifying the number of rows in the spreadsheet, see [Specifying the Number of Rows](#).
6. Choose the Apply button.
7. If your spreadsheet is not bound to a database, you must indicate the number of rows loaded in the [QueryData](#) event. After each data request, set the value of the *RowsLoaded* parameter to the number of rows that have been loaded.

[Print](#)

[Copy](#)

[Close](#)

### To customize virtual mode

#### VBX

1. Set the [VirtualMode](#) property to True to turn on virtual mode.
2. Specify the number of rows read into the virtual buffer at a time using the [VirtualRows](#) property.  
If the spreadsheet can display more rows at a time than the number of rows specified, the value of the VirtualRows property is reset to the number of rows the spreadsheet can display.
3. Set the [VirtualOverlap](#) property to specify the total number of previously retrieved rows that are retained in a buffer.
4. Set the [VirtualScrollBuffer](#) property to True to have the vertical scroll box reflect only the number of rows in the virtual buffer rather than the total number of rows.
5. If you set the VirtualScrollBuffer property to False or if you are displaying the row header column, use the [VirtualMaxRows](#) property to provide the spreadsheet with the number of rows in the spreadsheet or bound database.  
Providing the number of rows helps the vertical scroll bar more accurately reflect the current position in the spreadsheet and permits the row header column to accurately number the rows. For more information about specifying the number of rows in the spreadsheet, see [Specifying the Number of Rows](#).
6. If your spreadsheet is not bound to a database, you must indicate the number of rows loaded in the [QueryData](#) event. After each data request, set the value of the *RowsLoaded* parameter to the number of rows that have been loaded.

## Displaying the Special Vertical Scroll Bar

When using virtual mode, the vertical scroll bar does not always accurately represent the current location in the data. Also, the scroll bar can be slow to respond to changes in location.

Instead of displaying the default vertical scroll bar, you can display a special vertical scroll bar. The special scroll bar does not contain a scroll box. Instead, it displays scroll arrows that let you move to the first or last page, up or down a row, and up or down a page. The Spread control can display all or some of the scroll arrows.

- [OCX Instructions](#)
- [VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To display the special vertical scroll bar**

OCX

1. On the [Display](#) property page under Scroll Bars,
  - a. Select Both or Vertical from the Display drop-down list box.
  - b. Choose the Apply button.
2. On the [Virtual Mode](#) property page under Vertical Scroll Bar,
  - a. Select the Use Special Scroll Bar check box.
  - b. Specify which scroll arrows appear on the special scroll bar by selecting all, none, or some of the available check boxes.
  - c. Choose the Apply button.

[Print](#)

[Copy](#)

[Close](#)

**To display the special vertical scroll bar**

VBX

1. Set the [ScrollBars](#) property to 2 (Vertical) or 3 (Both).
2. Set the [VScrollSpecial](#) property to True.
3. At run time, use the [VScrollSpecialType](#) property to specify which scroll arrows are displayed.



## Working with Spreadsheets and Data

- [Opening an Existing Spreadsheet](#)
- [Saving a Spreadsheet](#)
- [Resetting a Spreadsheet to its Default Settings](#)
- [Printing a Spreadsheet](#)
- [Adding Data to a Spreadsheet](#)
- [Retrieving Data from a Spreadsheet](#)
- [Copying Data](#)
- [Moving Data](#)
- [Swapping Data](#)
- [Clearing Data](#)
- [Deleting Data](#)
- [Sorting Data](#)

## Opening an Existing Spreadsheet

You can load an existing spreadsheet that is in either a binary file or a tab-delimited file.

Binary files can contain all spreadsheet data and formatting or just spreadsheet data. Binary files are in a proprietary format that only the Spread control or the Spread Designer can read. For information about opening an existing spreadsheet using the Spread Designer, see [Opening a Spreadsheet in the Spread Designer](#).

Tab-delimited files contain spreadsheet data separated by tabs and carriage returns. Tab-delimited files can be opened, modified, and saved using any standard text editor.

By default, spreadsheet files created using Spread or the Spread Designer have the extension .SS2 (binary) or .TB2 (tab-delimited). However, you can use any extension.

When you create a spreadsheet, you can save that spreadsheet to either a binary file or a tab-delimited file. For instructions, see [Saving a Spreadsheet](#).

- [Loading from a binary file](#)
- [Loading from a tab-delimited file](#)

[Print](#)

[Copy](#)

[Close](#)

**To load an existing spreadsheet from a binary data file**

OCX

Call the [LoadFromFile](#) method.

VBX

Call the [SpreadLoadFromFile](#) function.

[Print](#)

[Copy](#)

[Close](#)

**To load an existing spreadsheet from a tab-delimited data file**

OCX

Call the [LoadTabFile](#) method.

VBX

Call the [SpreadLoadTabFile](#) function.

## Saving a Spreadsheet

You can save an entire spreadsheet (both data and formatting) or save only the spreadsheet data to a binary file. Saving only the spreadsheet data creates a much smaller file; however, no formatting information is saved.

You can also save the spreadsheet to a tab-delimited file.

Binary files can contain all spreadsheet data and formatting or just spreadsheet data. Binary files are in a proprietary format that only the Spread control or the Spread Designer can read. For information about saving a spreadsheet using the Spread Designer, see [Saving and Applying Changes](#).

Tab-delimited files contain spreadsheet data separated by tabs and carriage returns. Tab-delimited files can be opened, modified, and saved using any standard text editor.

By default, spreadsheet files created using Spread or the Spread Designer have the extension .SS2 (binary) or .TB2 (tab-delimited). However, you can use any extension.

- [Saving to a binary file](#)
- [Saving to a tab-delimited file](#)

[Print](#)

[Copy](#)

[Close](#)

**To save spreadsheet data or formatting and data to a binary file**

OCX

Call the [SaveToFile](#) method, and set the parameters as follows:

1. Set the value of the *FileName* parameter to the path and filename of the spreadsheet file you want to save.
2. Set the value of the *DataOnly* parameter to one of the following:
  - False to specify that both data and formatting be saved
  - True to specify that only data be saved

VBX

Call the [SpreadSaveToFile](#) function, and set the parameters as follows:

1. Set the value of the *FileName* parameter to the path and filename of the spreadsheet file you want to save.
2. Set the value of the *DataOnly* parameter to one of the following:
  - False to specify that both data and formatting be saved
  - True to specify that only data be saved

[Print](#)

[Copy](#)

[Close](#)

**To save a spreadsheet to a tab-delimited file**

OCX

Call the [SaveTabFile](#) method.

VBX

Call the [SpreadSaveTabFile](#) function.

## Resetting a Spreadsheet to its Default Settings

You can reset a spreadsheet to its default settings. Resetting a spreadsheet returns the spreadsheet to its initial state (prior to any design-time or run-time changes). Resetting a spreadsheet clears data, resets colors, and returns all cells to the default cell type (edit). You might want to reset the spreadsheet when you let the user choose the New choice from a File menu.

- [OCX, VBX Instructions](#)



[Print](#)

[Copy](#)

[Close](#)

**To reset a spreadsheet to its default settings**

OCX, VBX

At run time, set the [Action](#) property to 28 (Reset).

## Printing a Spreadsheet

You can print your spreadsheet and use a variety of options to customize printing. By default, spreadsheets are printed one page at a time, from top to bottom, then left to right. Spreadsheets are printed to the current default printer selected in your Windows environment. You can print spreadsheets on any Windows-supported printer.

You can simply print your spreadsheet, or you can specify print options. You can set print page margins, select the page orientation, and specify whether to print the spreadsheet border, column and row headers, grid lines, colors, and shadow effect. You can also specify header and footer text to appear on printed spreadsheet pages.

You can print a selected portion of a spreadsheet by specifying either a range of pages or a block of cells to print. Alternatively, you can print only the spreadsheet columns and rows that contain data. You can also specify page breaks before selected columns or rows.

For detailed instructions, see the following topics:

- [Sending a Spreadsheet to a Printer](#)
- [Specifying Print Options](#)
- [Printing the Portion of the Spreadsheet with Data](#)
- [Specifying a Print Range](#)
- [Specifying Print Page Breaks](#)
- [Adding Header and Footer Text to Printed Pages](#)

## Sending a Spreadsheet to a Printer

- [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To send a spreadsheet to a printer

OCX, VBX

1. Prepare the spreadsheet for printing by specifying print options and the portion of the spreadsheet to print.
  2. At run time, set the [Action](#) property to either of the following settings:
    - Use setting 13 (Print) to print the spreadsheet using the current print property settings.
    - Use setting 32 (Smart Print) to use Smart Printing provided by the spreadsheet.  
The Smart Print option determines the best orientation for printing the spreadsheet. It also scales the spreadsheet as appropriate to fit it on the page.
- Note** Do not use the Action property setting 32 (Smart Print) to print a spreadsheet that is using virtual mode. Use setting 13 (Print) instead.

## Specifying Print Options

Before printing a spreadsheet, you might want to specify print options such as margins and page orientation.

- [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

### To specify print options

OCX, VBX

At run time, customize any of the following print options by setting the corresponding property or properties:

1. Specify the page orientation used to print the spreadsheet with the [PrintOrientation](#) property.
2. Specify the size of the margins in twips using the [PrintMarginLeft](#), [PrintMarginRight](#), [PrintMarginTop](#), and [PrintMarginBottom](#) properties.
3. Specify whether to print the spreadsheet border with the [PrintBorder](#) property.
4. Specify whether to print the spreadsheet column header row and row header column with the [PrintColHeaders](#) and [PrintRowHeaders](#) properties.
5. Specify whether to print the shadow effect within the spreadsheet column header row and row header column with the [PrintShadows](#) property.

Note that if you set the [PrintColHeaders](#) and [PrintRowHeaders](#) properties to False in Step 4, the [PrintShadows](#) property setting will have no effect.

6. Specify whether to print the spreadsheet grid lines with the [PrintGrid](#) property.
7. Specify whether to print the colors in the spreadsheet as they appear on the screen with the [PrintColor](#) property.
8. Specify the print job name to display in the Print Manager when printing the spreadsheet with the [PrintJobName](#) property.
9. Specify the text to display in an abort dialog box during printing with the [PrintAbortMsg](#) property.

## Printing the Portion of the Spreadsheet with Data

You can specify that only the spreadsheet columns and rows that contain data are printed.

You can also set a specific block of cells or range of pages to print. For more information about setting a print range, see [Specifying a Print Range](#).

**Note** When printing a spreadsheet that is using virtual mode, set the [PrintUseDataMax](#) property to False.

— [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To print the portion of the spreadsheet with data**

OCX, VBX

At run time, set the [PrintUseDataMax](#) property to True.



## Specifying a Print Range

You can specify either a block of cells or a range of pages as a print range.

You can also specify that only the spreadsheet columns and rows that contain data are printed. For more information, see [Printing the Portion of the Spreadsheet with Data](#).

- [Specifying a block of cells as a print range](#)
- [Specifying a range of pages as a print range](#)

[Print](#)

[Copy](#)

[Close](#)

**To specify a block of cells as a print range**

OCX, VBX

At run time,

1. Set the [PrintUseDataMax](#) property to False.
2. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties to specify the block of cells you want to print.
3. Set the [PrintType](#) property to 1 (Cell Range).

[Print](#)

[Copy](#)

[Close](#)

**To specify a range of pages as a print range**

OCX, VBX

At run time,

1. Set the [PrintUseDataMax](#) property to False.
2. Specify the range of pages you want to print with the [PrintPageStart](#) and [PrintPageEnd](#) properties.
3. Set the [PrintType](#) property to 3 (Page Range).

## Specifying Print Page Breaks

You can set a print page break before a specified column or row. Page breaks are not displayed on the screen, but force page breaks when you print the spreadsheet. A column page break occurs to the left of the specified column; a row page break occurs above the specified row.

- [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To specify a print page break before a column or row**

OCX, VBX

1. If you want to specify a print page break before a column, at run time,
  - a. Specify the column with the [Col](#) property.
  - b. Set the [ColPageBreak](#) property to True.
2. If you want to specify a print page break before a row, at run time,
  - a. Specify the row with the [Row](#) property.
  - b. Set the [RowPageBreak](#) property to True.

## Adding Header and Footer Text to Printed Pages

You can specify header and footer text to appear on printed spreadsheet pages.

— [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To add header or footer text to printed spreadsheet pages**

OCX, VBX

At run time, provide the header or footer text using the [PrintHeader](#) and [PrintFooter](#) properties.

You can use control characters to specify header and footer format and content. For more information, refer to the [PrintHeader](#) and [PrintFooter](#) properties.

## Adding Data to a Spreadsheet

You can work with both formatted and unformatted data in the spreadsheet, depending on your needs.

Formatted data usually includes information that denotes the context of the data. Unformatted data does not include additional information and might require a specific format to convey meaning. For example, formatted data from a float cell can include currency and separator characters to indicate monetary value, as in \$1,025.34. Unformatted data from the same float cell does not include the currency and separator characters, as in 1025.34.

The following table lists the OCX/VBX properties, methods, and functions used to add formatted and unformatted data to the spreadsheet.

Type of Data	Spreadsheet Area	OCX/VBX Properties	OCX/VBX Methods or Functions
Formatted	A cell, a column, a row, or the entire spreadsheet	<a href="#">Text</a>	<a href="#">SetText</a>
	A block of cells	<a href="#">Clip</a> , <a href="#">Text</a>	
Unformatted	A cell, a column, a row, or the entire spreadsheet	<a href="#">Value</a>	
	A block of cells	<a href="#">ClipValue</a> , <a href="#">Value</a>	

The [Clip](#) and [ClipValue](#) properties work with ranges of cells, with tab characters separating columns and carriage return characters separating rows.

When working with formatted data, note that for date, edit, float, integer, PIC, static text, time, and combo box cells, a formatted data string is equivalent to the text representation being used on the screen. For button cells, a formatted data string is '0' for the up state and '1' for the down state. For check box cells, a formatted data string is '0' for the unchecked state and '1' for the checked state.

When working with unformatted data, note that date cells use "MMDDYYYY" notation. Time cells use an "HHMMSS" notation, where "HH" is in 24-hour format. Float cells remove any currency and separator characters. Combo box cells use the index (base zero) of the item in the list. PIC cells remove all mask characters.

Edit, integer, static text, button, check box, and owner-drawn cells use the same representation for both formatted and unformatted string data. The picture cell type has no text representation.

- [Adding formatted data](#)
- [Adding unformatted data](#)



## Adding Formatted Data

You can add formatted data to a specified spreadsheet cell, a column, a row, a block of cells, or the entire spreadsheet.

Supply data in the format in which it is to be displayed. For more information, refer to the [Text](#) property.

- [Adding data to a cell, a column, a row, or the entire spreadsheet](#)
- [Adding data to a block of cells](#)

[Print](#)

[Copy](#)

[Close](#)

### To add formatted data to a cell, a column, a row, or the entire spreadsheet

#### OCX

Either use the [Text](#) property or call the [SetText](#) method. Note that calling the SetText method is faster than using the Text property.

1. If you are using the [Text](#) property, at run time,
  - a. Set the [Col](#) and [Row](#) properties as appropriate to specify the spreadsheet area to which you want to add data.
  - b. Set the [Text](#) property to the data you want to add.
2. If you are using the method, call the [SetText](#) method.

#### VBX

Either use the [Text](#) property or call the [SpreadSetText](#) function. Note that calling the SpreadSetText function is faster than using the Text property.

1. If you are using the [Text](#) property, at run time,
  - a. Set the [Col](#) and [Row](#) properties as appropriate to specify the spreadsheet area to which you want to add data.
  - b. Set the [Text](#) property to the data you want to add.
2. If you are using the function, call the [SpreadSetText](#) function.

[Print](#)

[Copy](#)

[Close](#)

### To add formatted data to a block of cells

OCX, VBX

At run time,

1. Specify the block of cells to which you want to add data by setting the [Col](#), [Row](#), [Col2](#), and [Row2](#) properties.  
You do not need to set the [BlockMode](#) property.
2. Set the [Clip](#) property to the string of data you want to add.  
Separate data for each column with a tab character; separate each row with a carriage return character.

## Adding Unformatted Data

You can add unformatted data to a specified spreadsheet cell, a column, a row, a block of cells, or the entire spreadsheet.

For information about the cell types that can contain unformatted data and how unformatted data is interpreted by the cell, refer to the [Value](#) property.

- [Adding data to a cell, a column, a row, or the entire spreadsheet](#)
- [Adding data to a block of cells](#)

[Print](#)

[Copy](#)

[Close](#)

**To add unformatted data to a cell, a column, a row, or the entire spreadsheet**

OCX, VBX

At run time,

1. Set the [Col](#) and [Row](#) properties as appropriate to specify the spreadsheet area to which you want to add data.
2. Set the [Value](#) property to the data you want to add.

[Print](#)

[Copy](#)

[Close](#)

### To add unformatted data to a block of cells

OCX, VBX

At run time,

1. Set the [Col](#), [Row](#), [Col2](#), and [Row2](#) properties to specify the block of cells to which you want to add data.  
You do not need to set the [BlockMode](#) property.
2. Set the [ClipValue](#) property to the data you want to add.  
Separate data for each column with a tab character; separate each row with a carriage return character.

## Retrieving Data from a Spreadsheet

You can retrieve either formatted or unformatted data from a specified spreadsheet cell or block of cells.

The following table lists the OCX/VBX properties, methods, and functions used to retrieve formatted and unformatted data from the spreadsheet.

Type of Data	Spreadsheet Area	OCX/VBX Properties	OCX/VBX Methods or Functions
Formatted	A cell, a column, a row, or the entire spreadsheet	<a href="#">Text</a>	<a href="#">GetText</a>
	A block of cells	<a href="#">Clip</a>	
Unformatted	A cell, a column, a row, or the entire spreadsheet	<a href="#">Value</a>	
	A block of cells	<a href="#">ClipValue</a>	

For more information about formatted versus unformatted data, see [Adding Data to a Spreadsheet](#).

- [Retrieving formatted data](#)
- [Retrieving unformatted data](#)

## Retrieving Formatted Data

You can retrieve formatted data from a specified spreadsheet cell, a column, a row, a block of cells, or the entire spreadsheet.

- [Retrieving data from a cell, a column, a row, or the entire spreadsheet](#)
- [Retrieving data from a block of cells](#)



[Print](#)

[Copy](#)

[Close](#)

**To retrieve formatted data from a cell, a column, a row, or the entire spreadsheet**

OCX

Either use the [Text](#) property or call the [GetText](#) method. Note that using the GetText method is faster than using the Text property.

1. If you are using the [Text](#) property, at run time,
  - a. Set the [Col](#) and [Row](#) properties as appropriate to specify the spreadsheet area from which you want to retrieve data.
  - b. Retrieve the data using the [Text](#) property.
2. If you are using the method, call the [GetText](#) method.

VBX

Either use the [Text](#) property or call the [SpreadGetText](#) function. Note that using the SpreadGetText function is faster than using the Text property.

1. If you are using the [Text](#) property, at run time,
  - a. Set the [Col](#) and [Row](#) properties as appropriate to specify the spreadsheet area from which you want to retrieve data.
  - b. Retrieve the data using the [Text](#) property.
2. If you are using the function, call the [SpreadGetText](#) function.

[Print](#)

[Copy](#)

[Close](#)

**To retrieve formatted data from a block of cells**

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties to specify the block of cells from which you want to retrieve data.  
You do not need to set the [BlockMode](#) property.
2. Retrieve the data using the [Clip](#) property.  
Data for each column is separated with a tab character; each row is separated with a carriage return character.

## Retrieving Unformatted Data

You can retrieve unformatted data from a specified spreadsheet cell, a column, a row, a block of cells, or the entire spreadsheet.

- [Retrieving data from a cell, a column, a row, or the entire spreadsheet](#)
- [Retrieving data from a block of cells](#)

[Print](#)

[Copy](#)

[Close](#)

**To retrieve unformatted data from a cell, a column, a row, or the entire spreadsheet**

OCX, VBX

At run time,

1. Set the [Col](#) and [Row](#) properties as appropriate to specify the spreadsheet area from which you want to retrieve data.
2. Retrieve the data using the [Value](#) property.

[Print](#)

[Copy](#)

[Close](#)

**To retrieve unformatted data from a block of cells**

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties to specify the block of cells from which you want to retrieve data.  
You do not need to set the [BlockMode](#) property.
2. Retrieve the data using the [ClipValue](#) property.  
Data for each column is separated with a tab character; each row is separated with a carriage return character.

## Copying Data

You can copy data to and from cells using the familiar Clipboard method. Also, you can drag and drop data or use run-time property settings to copy a selected block of cells. The method you use will determine whether only the data or both data and formatting characteristics are copied.

When you copy data from one cell to another, the data from the cell you copied replaces the data in the cell you are pasting into. For example, if cell A1 contains the value 4, if you copy it and paste it into cell B3, the contents of cell B3 are replaced by the value 4. Also, when you copy data from one block of cells to another, the data from the copied block replaces the values in the block you paste into.

If the copy operation copies a block of cells and then pastes it at an overlapping location, the values of all the cells of the block you are pasting to are replaced with the values of the cells in the copied block, as shown in the following figure.

A1	B1	C1
A2	B2	C2
A3	B3	C3

*Copy cells A1 through B2*

A1	B1	C1
A2	A1	B1
A3	A2	B2

*Paste to cells B2 through C3  
(destination cell B2)*

- [Copying formatted data](#)
- [Copying unformatted data](#)

## Copying Formatted Data

You can copy formatted data from a specified block of cells to another block of cells in the same spreadsheet in code or at run time using the drag-and-drop method.

- [Copying data using code](#)
- [Copying data using the drag-and-drop method](#)

[Print](#)

[Copy](#)

[Close](#)

**To copy formatted data using code**

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties to specify the block of cells you want to copy.
2. Specify the upper-left cell of the destination block by setting the [DestCol](#) and [DestRow](#) properties.
3. Set the [Action](#) property to 19 (Copy Range).



[Print](#)

[Copy](#)

[Close](#)

### To copy data using the drag-and-drop method

#### OCX

1. On the [General](#) property page under Allow, select the Drag and Drop check box.
2. In your application, select the cell or cells you want to copy using the mouse.
3. Press the Ctrl key.
4. Position the mouse pointer over the edge of the selection so that the arrow pointer appears.
5. Press the left mouse button and drag the cell or cells.
6. Release the left mouse button when the mouse pointer is over the drop location.

#### VBX

1. Set the [AllowDragDrop](#) property to True.
2. In your application, select the cell or cells you want to copy using the mouse.
3. Press the Ctrl key.
4. Position the mouse pointer over the edge of the selection so that the arrow pointer appears.
5. Press the left mouse button and drag the cell or cells.
6. Release the left mouse button when the mouse pointer is over the drop location.

## Copying Unformatted Data

You can copy unformatted data with tab delimiters from the currently selected spreadsheet cell or block of cells to the Clipboard. Use the standard Clipboard keys to copy data to (Ctrl+C) and paste data from (Ctrl+V) the Clipboard.

When pasting data from the Clipboard, if a block of cells is selected, the block is replaced with the Clipboard contents. If the contents of the Clipboard are bigger than the selected block, the contents of the Clipboard are truncated at the edge of the block.

If no cells are selected, the contents of the Clipboard are pasted into the spreadsheet starting at the active cell. The contents of the cells to the right of and below the active cell are replaced by the contents of the Clipboard.

— [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To copy unformatted data from a cell or block of cells to the Clipboard**

OCX, VBX

At run time,

1. Specify the cell or block of cells from which you want to copy data by setting the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties.
2. Set the [Action](#) property to 2 (Select Block).
3. Set the Action property to 22 (Clipboard Copy).
4. Set the Col, Col2, Row, and Row2 properties as appropriate to specify the cell or block of cells into which you want to paste data.
5. Set the Action property to either 0 (Activate Cell) or 2 (Select Block).
6. Set the Action property to 24 (Clipboard Paste).

## Moving Data

You can move data to and from cells using the familiar Clipboard method of cutting and pasting data. Also, you can drag and drop data or use run-time property settings to move a selected block of cells. The method you use will determine whether only the data or both data and formatting characteristics are moved.

When you move data from one cell to another, the data from the first cell replaces the data in the cell you are moving data to. For example, if cell A1 contains the value 4, if you move the value and paste it into cell B3, the contents of cell B3 are replaced by the value 4.

When you move a block of data, the destination block is automatically sized based on the size of the block being moved. If data exists in the destination block, the existing data is replaced by the data in the block being moved.

If the move operation moves a block of cells to an overlapping location, the values of all the cells of the block you are moving to are replaced with the values of the cells in the moved block, as shown in the following figure.

A1	B1	C1
A2	B2	C2
A3	B3	C3

*Move cells A1 through B2*

		C1
	A1	B1
A3	A2	B2

*Replace cells B2 through C3  
(destination cell B2)*

When you move blocks of data, you can specify whether formulas are adjusted. For more information, see [Automatically Recalculating and Updating Formulas](#).

- [Moving formatted data](#)
- [Moving unformatted data](#)

## Moving Formatted Data

You can move formatted data from a specified block of cells to another block of cells in the same spreadsheet in code or at run time using the drag-and-drop method.

- [Moving data using code](#)
- [Moving data using the drag-and-drop method](#)

[Print](#)

[Copy](#)

[Close](#)

### To move formatted data using code

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties to specify the block of cells you want to move.
2. Specify the upper-left cell of the destination block by setting the the [DestCol](#) and [DestRow](#) properties.
3. Set the [Action](#) property to 20 (Move Range).

[Print](#)

[Copy](#)

[Close](#)

### To move data using the drag-and-drop method

#### OCX

1. On the [General](#) property page under Allow, select the Drag and Drop check box.
2. In your application, select the cell or cells you want to move using the mouse.
3. Position the mouse pointer over the edge of the selection so that the arrow pointer appears.
4. Press the left mouse button and drag the cell or cells.
5. Release the left mouse button when the mouse pointer is over the drop location.

#### VBX

1. Set the [AllowDragDrop](#) property to True.
2. In your application, select the cell or cells you want to move using the mouse.
3. Position the mouse pointer over the edge of the selection so that the arrow pointer appears.
4. Press the left mouse button and drag the cell or cells.
5. Release the left mouse button when the mouse pointer is over the drop location.

## Moving Unformatted Data

You can cut unformatted data from a selected spreadsheet cell or block of cells to the Clipboard, and paste unformatted data from the Clipboard into a selected cell or block of cells. Use the standard Clipboard keys to move data to (Ctrl+X) and paste data from (Ctrl+V) the Clipboard.

When pasting data from the Clipboard, if a block of cells is selected, the block is replaced with the Clipboard contents. If the contents of the Clipboard are bigger than the selected block, the contents of the Clipboard are truncated at the edge of the block.

If no cells are selected, the contents of the Clipboard are pasted into the spreadsheet starting at the active cell. The contents of the cells to the right of and below the active cell are replaced by the contents of the Clipboard.

— [QCX, VBX Instructions](#)



[Print](#)

[Copy](#)

[Close](#)

**To move unformatted data**

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate to specify the cell or block of cells from which you want to cut data.
2. Set the [Action](#) property to 2 (Select Block).
3. Set the Action property to 23 (Clipboard Cut).
4. Set the Col, Col2, Row, and Row2 properties as appropriate to specify the cell or block of cells into which you want to paste data.
5. Set the Action property to either 0 (Activate Cell) or 2 (Select Block).
6. Set the Action property to 24 (Clipboard Paste).

## Swapping Data

You can swap the contents of two cells or two blocks of cells.

When you swap data from one cell to another, the data in one cell becomes the data in the other cell, and vice versa. For example, if cell A1 contains the value 4 and cell B3 contains the value 6, if you swap the values of the cells, the value of cell A1 becomes 6 and the value of cell B3 becomes 4.

If you attempt to swap a block that is larger than the available block at the destination, the swap operation is not performed. For example, if you attempt to swap a block of four cells and specify the destination as a cell at the edge of the spreadsheet, the swap does not take place.

If the swap operation swaps overlapping blocks of cells, the destination block is swapped, and then the selected block is swapped, overwriting the destination block, as shown in the following figure.

A1	B1	C1
A2	B2	C2
A3	B3	C3

*Swap selected cells A1 through B2  
with cells B2 through C3*

B2	C2	C1
B3	A1	B1
A3	A2	B2

*Cell C3 is overwritten by cell A1*

When you swap blocks of data, you can specify whether formulas are adjusted. For more information, see [Automatically Recalculating and Updating Formulas](#).

— [OCX, VBX Instructions](#)

[Print](#)

[Copy](#)

[Close](#)

**To swap formatted data between two blocks of cells**

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties to specify the first block of cells you want to swap.
2. Specify the upper-left cell of the second block of cells you want to swap by setting the [DestCol](#) and [DestRow](#) properties.
3. Set the [Action](#) property to 21 (Swap Range).

## Clearing Data

You can clear both data and formatting information from a selected spreadsheet cell or block of cells, or clear only the data, leaving the formatting information, such as cell type, intact.

- [Clearing the data and formatting from a cell or block of cells](#)
- [Clearing only the data from a cell or block of cells](#)

[Print](#)

[Copy](#)

[Close](#)

**To clear the data and formatting from a cell or block of cells**

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate to specify the cell or block of cells from which you want to clear the data and formatting.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. Set the [Action](#) property to 3 (Clear).
4. If you selected a block of cells, set the BlockMode property back to False.

[Print](#)

[Copy](#)

[Close](#)

**To clear only the data from a cell or block of cells**

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties as appropriate to specify the cell or block of cells from which you want to clear data.
2. If you are selecting a block of cells, set the [BlockMode](#) property to True.
3. Set the [Action](#) property to 12 (Clear Text).
4. If you selected a block of cells, set the BlockMode property back to False.

## Deleting Data

You can delete spreadsheet data using several different methods. For more information, see the following topics:

- [Moving Data](#)
- [Clearing Data](#)
- [Deleting Columns and Rows](#)
- [Resetting a Spreadsheet to its Default Settings](#)

## Sorting Data

You can sort spreadsheet data using up to three sort keys.

**Note** You cannot sort a spreadsheet that is using virtual mode.

- [OCX\\_VBX\\_Instructions](#)



[Print](#)

[Copy](#)

[Close](#)

### To sort spreadsheet data

OCX, VBX

At run time,

1. Set the [Col](#), [Col2](#), [Row](#), and [Row2](#) properties to specify the block of cells you want to sort.
2. Specify whether to sort by columns or by rows with the [SortBy](#) property.
3. For each sort key,
  - a. Specify the column or row number and the sort key number (with the *Index* parameter) with the [SortKey](#) property.
  - b. Specify whether to sort in ascending or descending order with the [SortKeyOrder](#) property.
4. Set the [Action](#) property to 25 (Sort).

**Note** If you are using formulas and the FormulaSync property is set to True, formulas are *not* adjusted.

# Contents

## ***User's Guide***

### **Introduction**

[Introducing Spread](#)

[Using the Spread Designer](#)

[Using OCX Controls](#)

[Spread Control Features](#)

### **How-to Guide**

[Working with Spreadsheets and Data](#)

[Working with Cells](#)

[Customizing Columns and Rows](#)

[Customizing the Spreadsheet Display](#)

[Optimizing Your Spreadsheet](#)

[Binding the Spreadsheet to a Database](#)

## ***Reference Guide***

[Properties](#)

[Events](#)

[Functions](#)

