

**See Also**

[How the DataGrid handles data validation & error checking](#)

## **(About) Applies To**

[SSDBCombo](#)

[SSDBCommand](#)

[SSDBDropDown](#)

[SSDBData](#)

[SSDBGrid](#)

[SSDBOptSet](#)

## **(About) Property**

Applies To

### **Description**

Displays version information about the control.

### **Usage**

Click on the ellipses ('...') button next to the property text to activate the about dialog box.

### **Remarks**

This property is available only at design time. At runtime, it is available as a method called **AboutBox**.

## **(Custom) Property**

### **Applies To**

SSDBCombo, SSDBCommand, SSDBData, SSDBDropDown, SSDBGrid, SSDBOptSet

### **Description**

Displays the property pages for the control.

### **Usage**

Right-click on the object and choose 'Properties...' from the pop-up menu, or click on the ellipses ('...') button next to the property text to activate the property pages.

### **Remarks**

This property is available only at design time.

With this release of Data Widgets also comes the transition from accessing and manipulating grid rows by row number to using bookmarks. Row numbers are still used in Data Widgets 2.0, but their meaning has been redefined. There are now two types of row numbers, *Display* row numbers which indicate the row number as it appears in the current view, and *Absolute* row numbers which indicate the row number relative to the entire grid. Absolute row numbers are *only* used in AddItem mode, while display row numbers are used in bound, unbound, and AddItem modes of the grid.

## About SelBookmarks

The SelBookmarks collection represents a set of selected bookmark objects. Bookmarks are added to this collection whenever a user selects a row in the grid. If Multiselect is True, then each row selected will be added to the collection in the order in which the selection occurred. Order is never based on the displayed order. When a row is unselected, that row is then removed from the SelBookmarks collection. It is dangerous and not recommended to ever store the ordinal position of a row within this collection.

You can also add bookmarks to the SelBookmarks collection through code. The following example will add the first five rows to the collection:

```
Dim i as integer

SSDBGrid1.MoveFirst ' Position at the first row
For i = 0 to 4
    SSDBGrid1.SelBookmarks.Add SSDBGrid1.Bookmark
    SSDBGrid1.MoveNext
Next i
```

It is also easy to access the rows in the SelBookmarks collection without moving the current row position. For example, if there was a column in the grid called Amount and you wanted to add up all the rows that were selected to get a total, you could use the following code:

```
Dim nTotal as long
Dim nTotalSelRows as integer
Dim i as integer
Dim bkmrk as Variant ' Bookmarks are always defined as variants

nTotalSelRows = SSDBGrid1.SelBookmarks.Count

' In the following, get the bookmark of the selected rows

For i = 0 to nTotalSelRows
    bkmrk = SSDBGrid1.SelBookmarks(i)
    nTotal = nTotal + SSDBGrid1.Columns("Amount").CellValue(bkmrk)
Next i

Debug.Print "The total amount = " & Format(nTotal, "Currency")
```

## About StyleSets

To understand StyleSet objects and the StyleSets collection, you should become familiar with the concept of *collections*.

A StyleSet is an object that contains a set of visual properties. In Data Widgets, the Data Grid, Data Combo, and Data DropDown all make use of StyleSet objects.

## Creating StyleSets

It is possible to create style sets through the Grid Editor. It is also possible, and sometimes more appropriate, to create StyleSets through code.

In the case of the Data Grid, different StyleSets can be created and applied to specific columns, groups, and headers. Each of these StyleSets can in-turn be given characteristics which make one stand out from the other. For example, a "Loss" column of a grid containing financial data can have its **BackColor** property set to 'red', while a "Profit" column of a grid can have its **BackColor** property set to "green".

The following is an example of how a StyleSet may be set up:

1. The StyleSet is first added to the StyleSets Collection as follows:

```
SSDBGrid1.StyleSets.Add "Houston"
```

2. Once added, the properties of a StyleSet may be set as follows:

```
SSDBGrid1.StyleSets("Houston").BackColor = RGB(255,255,0)
```

## Applying StyleSets

Once the StyleSet is created, it can be applied to an object that has a **StyleSet Property**. The following code applies the 'Houston' StyleSet to the **Column** object of a Data Grid control:

```
SSDBGrid1.Columns(1).StyleSet = "Houston"
```

**Note** If a change is made to a StyleSet, it does not have to be reapplied to an object to take effect. However, the control may need to be redrawn by invoking the **Refresh** method.

## Achieving a 3D Look with the Data Combo

By setting just a few properties, you can quickly make your Data Combo have a 3D look to it. The following settings allow for a 3D look:

```
SSDBCombo1.BackColorEven = &H00C0C0C0& \ Gray  
SSDBCombo1.BackColorOdd = &H00C0C0C0& \ Gray  
SSDBCombo1.ForeColorEven = &H00000000& \ Black  
SSDBCombo1.ForeColorOdd = &H00000000& \ Black  
SSDBCombo1.DividerStyle = 3 \ Inset  
SSDBCombo1.DividerType = 3 \ Both
```

Au_ID	Author	Year	▲
1	Adams, Pat		
2	Adriaan, Merv		
3	Ageloff, Roy	1943	
4	And		
5	Antonovich Michael P.		
6	Arnott, Steven E.	21	
7	Antson, L. Joyce		
8	Ault, Michael R		▼



## **ActiveCell Applies To**

SSDBGrid

## ActiveCell Method

[See Also](#)

[Applies To](#)

### Description

Returns the active cell object.

### Syntax

*object* . **ActiveCell**

Part	Description
------	-------------

---

<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
---------------	---

### Remarks

The **ActiveCell** object is singular and not a member of a collection.

## **ActiveCell Method See Also**

ActiveCell Object

## ActiveCell Object

[See Also](#)

[Applies To](#)

### Description

An ActiveCell object represents the active cell in the grid.

### Properties

---

<a href="#">Height</a>	<a href="#">SelText</a>	<a href="#">Value</a>
<a href="#">Left</a>	<a href="#">StyleSet</a>	<a href="#">Width</a>
<a href="#">SelLength</a>	<a href="#">Text</a>	
<a href="#">SelStart</a>	<a href="#">Top</a>	

### Remarks

The ActiveCell is not a member of any collection, and is a singular object. The **Left**, **Top**, **Width**, and **Height** properties are read-only for this object. The numbers exposed on this object are in screen coordinates.

## **ActiveCell Object See Also**

ActiveCell Method

## ActiveRowStyleSet Property

[See Also](#)

[Applies To](#)

### Description

Returns or sets the name of the **ActiveRowStyleSet** in the **StyleSets** collection. The **ActiveRowStyleSet** determines the style of the active row.

### Syntax

*object* . **ActiveRowStyleSet**[= *text* ]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>text</i>	A string expression that evaluates to the name of the <b>ActiveRowStyleSet</b> .

### Remarks

This property determines the StyleSet to be used for the active row of the control. Note that the **StyleSet** specified must be in the **StyleSets** collection. StyleSets will override each other based on the following hierarchy:

#### **Data Area:**

<b>ActiveCell.StyleSet</b>	(overrides all below)
<b>Control.ActiveRowStyleSet</b>	(overrides all below)
<b>Column.CellStyleSet</b>	(overrides all below)
<b>Column.StyleSet</b>	(overrides all below)
<b>Group.StyleSet</b>	(overrides all below)
<b>Control.StyleSet</b>	

### Properties

---

[BackColor](#)      [Font](#)      [ForeColor](#)

## **ActiveRowStyleSet Property See Also**

[HeadStyleSet](#) property

[StyleSet](#) property

[StyleSet](#) object

[StyleSets](#) collection

## **ActiveRowStyleSet Applies To**

SSDBGrid



## Add Method

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Adds objects to a collection.

### Syntax (*SSDBData: Bookmarks*)

*object* . **Add**(*bookmark* **As Variant**, *bookstring* **As String**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>bookmark</i>	A variant specifying the bookmark number to add.
<i>bookstring</i>	A string specifying the text displayed when the bookmark is referenced.

### Syntax (*SSDBGrid: Columns/Groups*)

*object* . **Add**(*Index* **As Integer**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>index</i>	An integer specifying where the column or group is inserted in a collection.

### Syntax (*SSDBGrid: SelBookmarks*)

*object* . **Add**(*bookmark* **As Variant**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>bookmark</i>	A variant specifying the bookmark number to add.

### Syntax (*SSDBOptSet:Buttons*)

*object* . **Add**([*ButtonsToAdd* **As Variant**])

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>ButtonsToAdd</i>	Total number of objects to add.

### Syntax (*StyleSets*)

*object* . **Add**(*name* **As String**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>name</i>	A string expression specifying the name of the StyleSet to add.

### **Remarks**

For the **Buttons** collection, if you do not specify a value for *ButtonsToAdd*, only one button will be added.

For the **Columns** and **Groups** collections, you can insert either between two existing collections.

For example, if you have three columns and you want to add a column between the second and third (Columns(1) and Columns(2)), your code would look like:

```
SSDBGrid1.Columns.Add(2)
```

## **Add Method Applies To**

Bookmarks collection

Buttons collection

Columns collection

Groups collection

SelBookmarks collection

StyleSets collection

## **Add Method See Also**

Bookmarks collection

Buttons collection

Columns collection

Groups collection

StyleSets collection

Count property

Remove method

RemoveAll method

## AddItem Method

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Adds a string to an AddItem grid.

### Syntax

*object*. **AddItem**(*Item* **As String**, [*index* **As Integer**])

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>item</i>	A string expression specifying the data to add.
<i>index</i>	An integer expression specifying the row to add the string to.

### Remarks

When no index is specified, the string is added at the end of the grid.

## AddItem Method (Column Object)

[See Also](#)

[Applies To:](#)

[Example](#)

### Description

Adds a string to the column's combo box.

### Syntax

*object*. **AddItem**(*Item* **As String**, [*index* **As Integer**])

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>item</i>	A string expression specifying the data to add.
<i>index</i>	An integer expression specifying the position in the combo box to add the data.

### Remarks

This method is only valid for columns with the style set to combo box. If you do not specify an index, the string is appended to the list.

## **AddItem Method (Column Object) Applies To**

Column Object

## **AddItem Method (Column Object) See Also**

Style

Columns collection



## **AddItem Method Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **AddItem Method See Also**

[DataMode](#)

[FieldDelimiter](#)

[FieldSeparator](#)

[RemoveItem](#) method

## AddItemBookmark Method

[See Also](#)

[Applies To](#)

### Description

Returns the AddItem bookmark for a given absolute row number.

### Syntax

*object* . **AddItemBookmark**(*RowIndex* **As Long**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>RowIndex</i>	A numeric expression specifying the absolute row number.

### Remarks

In bound mode, you are able to access bookmarks through the data control. This method gives the programmer access to the bookmarks for AddItem mode.

## **AddItemBookmark Method Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

**AddItemBookmark Method See Also**

[AddItemRowIndex](#)

## AddItemRowIndex Method

[See Also](#)

[Applies To](#)

### Description

Returns the AddItem row number for a given bookmark.

### Syntax

*object* . **AddItemRowIndex**(*Bookmark* **As Variant**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Bookmark</i>	A variant expression specifying the bookmark.

### Remarks

In bound mode, you are able to access bookmarks through the data control. This method gives the programmer access to the row number based on a bookmark for AddItem mode.

## **AddItemRowIndex Method Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **AddItemRowIndex Method See Also**

[AddItemBookmark](#)



## **Adding An AddItem Grid See Also**

[Performance Tuning](#)

## **Adding An Unbound Data Combo See Also**

[Adding An Unbound Data DropDown](#)

[Adding An Unbound Data Grid](#)

[Performance Tuning](#)

## **Adding An Unbound Data DropDown See Also**

[Adding An Unbound Data Combo](#)

[Adding An Unbound Data Grid](#)

[Performance Tuning](#)

## **Adding An Unbound Data Grid See Also**

[Adding An Unbound Data DropDown](#)

[Adding An Unbound Data Combo](#)

[Performance Tuning](#)

[◀ Back](#)

## **Adding Pictures to Cells**

**How do you want to apply the pictures to the cells?**

Apply pictures based on the contents of a cell

Apply the same picture to every cell in a column

## Adding a Bound Data Combo

The Data Combo relies on the host environment's standard data control to access database information.

### To use the Data Control with your application in Visual Basic:

1. Place two standard data controls on your form. One is used for the edit portion, the other is used for the list portion.
2. For both data controls, set the **DatabaseName** and **RecordSource** properties to point to a database and the table within the database.
3. Place a Data Combo control on your form.
4. Set the **DataSource** property of the Data Combo to point to the data control used for the edit portion. Set the **DataField** property to point to the field used.
5. Set the **DataSourceList** property of the Data Combo to point to the data control used for the list portion. Set the **DataFieldList** property to point to the field used.

## Adding a Bound Data DropDown

Much like a Data Combo, the field in the cell is related to the list in the Data DropDown.

### To use the Data DropDown in a Data Grid:

1. Place two standard data controls on your form. One is used for the data grid, the other is used for the data combo.
2. For both data controls, set the **DatabaseName** and **RecordSource** properties to point to a database and the table within the database.
3. Place a Data Grid control on the form and bind it to the first data control.
4. Place a Data DropDown control on the form. The location of the Data DropDown is unimportant since it is invisible at runtime.
5. Set the **DataSource** property of the Data DropDown to the second data control.
6. Set the **DataFieldList** property of the Data DropDown to the field you want used for the list.
7. Link the Data DropDown to the Data Grid by adding the following code in the **InitColumnProps** procedure of the Data Grid:

```
SSDBGrid1.Columns(n).DropDownhWnd = SSDBDropDown1.hWnd
```

**Note** The instructions above assume that the Data Grid is also bound to a data control. It is possible to have a bound Data DropDown work in conjunction with an unbound Data Grid. If the Data Grid is unbound, you will only need one data control, the one used for the Data DropDown.

## Adding a Bound Data Grid to your application

The Data Grid makes use of the host environment's standard data control.

### To create a functional grid for your application in Visual Basic:

1. Add a Visual Basic Data Control to your form.
2. Set the **DatabaseName** and **RecordSource** properties in the data control.
3. Add a SSDBGrid Control to your form.
4. Set the **DataSource** property in the SSDBGrid control to the data control (i.e., Data1).

Your grid is now aware of the database associated with the data control. At this point, you can use the [Grid Editor](#) to design a grid format.



## Adding a Data Command Button

The Data Command button only works when bound to a data control.

### To use the Data Command button with your application in Visual Basic:

1. Place a standard data control on your form.
2. Set the **DatabaseName** and **RecordSource** properties to point to a database and the table within the database.
3. Place the Data Command button on your form.
4. Set the **DataSource** property of the Data Command button to point to the data control you created in Step 1.
5. Set the **DatabaseAction** property of the Data Command button to perform the action you want.

[◀ Back](#)

## Adding a computed column to the DataGrid

There may be cases in which you have a series of values in a record and you want to display a computed value in an unbound column. An example of this would be an Orders database where you have Quantity and Unit Price columns, and you wish to display a Total Price that is the product of Quantity and Unit Price. Such a value does not need to be stored in the database; it can be computed and displayed "on the fly."

Take the following steps to display computed values in an unbound column:

1. Add an unbound column to the grid.

[Click here](#) to see how to do this.

2. In the **RowLoaded** event, perform the desired calculation and assign it to the current cell in the unbound column.

[Click here](#) to see how to do this.

## Adding an AddItem Grid to Your Application

### See Also

In AddItem mode, you can add as many rows of data as you want, at any time during operation. This data is accessible as if the grid was bound (i.e., when you scroll, the next row of data is displayed automatically). Instead of a data control managing the flow of data, the grid does.

This mode operates similarly to the Visual Basic list box, but has all the features and power of the SSDBGrid. You can use the Grid Editor to help create the AddItem grid, or you can manually specify the properties.

Wherever possible, the grid in AddItem mode has the same functionality as a grid in bound mode, and most programmatic statements are the same.

The uses of this mode are virtually endless. One of its most useful features is being able to fill the grid with information without the need for a database. The AddItem mode is much better on system resources because it does not require the overhead of a Data Control. AddItem mode is best used for small lists that are easily maintained.

### **To create a an AddItem grid for your application:**

1. Add a SSDBGrid control to your form.
2. Set the **DataMode** property to 2 (AddItem mode).
3. Specify the number of columns to use by setting the **Cols** property.
4. If you want to change the **FieldDelimiter** or **FieldSeperator** properties from their defaults, specify them now.
5. Specify code in the **InitColumnProps** event of the grid so that items are added when the grid first appears.

The following example demonstrates how **InitColumnProps** can be used to fill an AddItem grid:

```
Sub SSDBGrid1_InitColumnProps
Dim I As Integer
  For I = 0 to 32
    SSDBGrid1.AddItem "Hello" + CHR$(9) + "World"
  Next I
End Sub
```

## Adding an Unbound Data Combo

### See Also

the Data Combo has two portions that it retrieves data for, with the ability to bind each part to different data sources. You can also configure the Data Combo to have either or both portions unbound, in which case, you will need to supply the data yourself.

When the edit portion of the Data Combo is unbound, you need to initially supply the field value yourself via the **Text** property which contains the value of the data in the edit portion of the Data Combo. When the user clicks on the dropdown button, the list portion will automatically update the **Text** property and the contents of the edit portion if the user selects a value, much like a standard combo box. The functionality of the Data Combo in unbound mode is identical to the Data Grid in unbound mode.

You can also set the Data Combo to AddItem mode, following the same guidelines used for the Data Grid in AddItem mode.

## **Adding an Unbound Data DropDown**

### See Also

The functionality of the Data DropDown in unbound mode is identical to the [Data Grid in unbound mode](#).

You can also set the Data DropDown to AddItem mode, following the same guidelines used for the [Data Grid when in this mode](#).

## Adding an Unbound Data Grid to your application

### See Also

The primary use of the Data Grid is to manage the display and entry of data into a record set of the bound data control. Because a database may contain an unlimited amount of data, the Data Grid has to manage the data in a virtual fashion, meaning that it only reads in as much data as it needs to display information on the screen.

Another important feature of the Data Grid is its ability to perform as an unbound control. Unbound mode is most useful when you need to handle data that the host environment's standard data control cannot. The only difference between bound and unbound mode is how the data is handled coming into the grid and going out of the grid.

The unbound grid sends cues in the form of events notifying you when it needs a response. When it needs more data, it fires the **UnboundReadData** event, likewise, when it needs to save data, it fires the **UnboundWriteData** event. Your primary responsibility in unbound mode is to supply the grid with data when it requests it, and to store data when it sends it.

### **To create an unbound grid for your application in Visual Basic:**

1. Add a SSDBGrid Control to your form.
2. Set the **DataMode** to '1 - Unbound'.
3. Place code in the **UnboundReadData** event of the Data Grid that extracts data from your data source.
4. Place code in the **UnboundWriteData** event of the Data Grid that writes modified data back to your data source.
5. Place code in the **UnboundAddData** event of the Data Grid that appends a new row to your data source.
6. Place code in the **UnboundDeleteRow** event of the Data Grid that deletes a row from your data source.

Your unbound data grid is now ready for use.

## Adding an unbound column to a grid through code

Adding unbound columns to a grid control is accomplished using the **Add** method of the Columns collection. You must specify the index of the new column when invoking the method. Once you have added the new column, you can immediately begin setting properties, such as **Caption** and **DataType**.

If you want the unbound column to be part of the grid as soon as it appears, add the code to the **InitColumnProps** event. Alternatively, you could add the column in response to some other event, such as the user clicking a command button.

The following code adds a new column to the right side of a DataGrid, based on the number of columns already in the grid.

```
(General) (declarations)
    Dim iNewCol As Integer

Private Sub SSDBGrid1_InitColumnProps()
    iNewCol = SSDBGrid1.Columns.Count
    SSDBGrid1.Columns.Add iNewCol
    SSDBGrid1.Columns(iNewCol).Caption = "Total Price"
    SSDBGrid1.Columns(iNewCol).DataType = 4
End Sub
```

Often, you want to add an unbound column to display some value computed from other columns in the grid. The following code shows an example of how to perform such a calculation. For the purposes of this example, Columns(2) contains a Unit Price, and Columns(3) contains a Quantity. The code multiplies the two values to produce a total price, which is then displayed in the unbound column:

```
Private Sub SSDBGrid1_RowLoaded(ByVal Bookmark As Variant)
    Dim nTotal As Single

    nTotal = (SSDBGrid1.Columns(2).CellValue(Bookmark) *
SSDBGrid1.Columns(3).CellValue(Bookmark))
    SSDBGrid1.Columns(iNewCol).DataType = 3
    SSDBGrid1.Columns(iNewCol).Value = Format(nTotal, "Currency")
End Sub
```





[◀ Back](#)

## **Adding an unbound column using the Grid Editor**

Take the following steps to use the Grid Editor at design time to set up a Data Grid, Data Combo or Data DropDown with one or more unbound columns.

- 1.** Place the control on the form and specify a valid data source for it.
- 2.** Right-click on the control to bring up the context menu. Select "Properties..." from the menu. The property pages will appear.
- 3.** Click on the Columns tab of the property pages. This displays the Grid Editor.
- 4.** Click on the "Fields..." button. A dialog will appear with the available fields from the data source you specified. Select the fields you want to appear in the grid, using the CTRL key and the mouse to select / deselect individual fields. Click OK when finished selecting fields.
- 5.** The Grid Editor will now display a preview of the grid with the fields you have selected. So far all the columns are bound columns.
- 6.** Click the "Add Column" button. A dialog will appear for you to specify the name of the new column. Enter the name you wish to use for your unbound column and click OK.
- 7.** The new, unbound column will appear with the name you specified in the caption. New columns appear to the right of the existing columns in the grid. You can then change any of the column's attributes using the Grid Editor controls, or move it to a different position within the grid by dragging it with the mouse.

## Adding the DataOptionSet

Option buttons for the DataOptionSet can be created at either design or runtime. Once you have placed the control on the form, all you need to do is set properties that define the values for your DataOptionSet.

### To use the DataOptionSet with your application in Visual Basic:

1. Place a standard data control on your form.
2. Set the **DatabaseName** and **RecordSource** properties to point to a database and the table within the database.
3. Place the DataOptionSet on your form.
4. Set the **DataSource** property of the DataOptionSet to point to the data control you created in Step 1.
5. Set the **DataField** property of the DataOptionSet so that it points to the field to work with.

The DataOptionSet has been added to your form, but you must create buttons at design time or create buttons at run time in order for the control to be useful.

## Adding the Enhanced Data Control

Adding the Enhanced Data Control to your form is quite simple. Remember that the EDC works in *conjunction* with the standard data control, not without it.

### To use the Enhanced Data Control with your application:

1. Place a standard data control on your form.
2. Set the **DatabaseName** and **RecordSource** properties to point to a database and the table within the database.
3. Place the Enhanced Data Control on your form.
4. Set the **DataSource** property of the EDC to point to the data control you created in Step 1.
5. Set the **DataField** property of the EDC to point to the database field you want the EDC bound to.

## **Additional Visual Basic 4.0 Samples**

[Applying Pictures to Cells](#)

[Totalling Values in a Grid Column](#)

[Creating a Total Query](#)

[Adding an Unbound Column to a Grid Through Code](#)

## AfterClick Event

[Applies To](#)

[Example](#)

### Description

Occurs immediately after the user has clicked the button and the database action has been performed.

### Syntax

**SSDBCommand:**

**Sub** control\_AfterClick ()

**SSDBData:**

**Sub** control\_AfterClick (ByVal *nPosition* As Integer)

The event parameters are:

#### Parameter Description

---

*nPosition* Integer indicating the area of the control being pointed to.

### Remarks

**AfterClick** gives the user an opportunity to perform an action after a database action has been performed.

The following values for *nPosition* apply to the Enhanced Data Control

Integer	Area being pointed to
1	Caption Area
2	Bevel Area
3	"First" Button
4	"Last" Button
5	"Previous Page" Button
6	"Next Page" Button
7	"Previous Record" Button
8	"Next Record" Button
9	"Add" Button
10	"Cancel" Button
11	"Update" Button
12	"Delete" Button
13	"Find Next" Button
14	"Find Previous" Button
15	"Find" Button
16	"Add Bookmark" Button
17	"Clear Bookmark" Button
18	"Goto Bookmark" Button

There are constants available for the settings of this parameter.

## **AfterClick Event Applies To**

SSDBCommand

SSDBData

## AfterColUpdate Event

[See Also](#)

[Applies To](#)

### Description

Occurs after data is moved from a cell in the grid to the control's copy buffer.

### Syntax

**Sub** control\_ **AfterColUpdate** (*ColIndex* **As Integer**)

The event parameters are:

### Parameter Description

---

*ColIndex* An integer expression that specifies the index of the column in which the data is moved from.

*Remarks* The AfterColUpdate event occurs only if the *Cancel* argument in the **BeforeColUpdate** event is not set to **True**.



## **AfterColUpdate Event Applies To**

SSDBGrid

## **AfterColUpdate Event See Also**

[AfterInsert](#) event

[AfterUpdate](#) event

[BeforeColUpdate](#) event

[BeforeDelete](#) event

[BeforeInsert](#) event

[BeforeUpdate](#) event

## AfterDelete Event

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Occurs after the user deletes a row.

### Syntax

**Sub** control\_ **AfterDelete** (*RtnDispErrMsg* **As Integer**)

The event parameters are:

#### **Parameter Description**

---

*RtnDispErrMsg* An integer expression that indicates if an error message box should be displayed.

## **AfterDelete Event Applies To**

SSDBGrid

## **AfterDelete Event See Also**

AfterColUpdate event

AfterInsert event

AfterUpdate event

BeforeColUpdate event

BeforeDelete event

BeforeInsert event

BeforeUpdate event

## AfterInsert Event

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Occurs after the user inserts a new row.

### Syntax

**Sub** control\_\*\*AfterInsert\*\* (*RtnDispErrMsg* **As Integer**)

The event parameters are:

#### **Parameter Description**

---

*RtnDispErrMsg* An integer expression that indicates if an error message box should be displayed.

## **AfterInsert Event Applies To**

SSDBGrid

## **AfterInsert Event See Also**

[AfterColUpdate](#) event

[AfterDelete](#) event

[AfterUpdate](#) event

[BeforeColUpdate](#) event

[BeforeDelete](#) event

[BeforeInsert](#) event

[BeforeUpdate](#) event



## AfterPosChanged Event

[See Also](#)

[Applies To](#)

### Description

Occurs just after a grid column changes position due to movement or swapping.

### Syntax

```
Sub control_AfterPosChanged (ByValWhatChanged As Integer, ByVal NewIndex As Integer )
```

The event parameters are:

Parameter	Description
<i>WhatChanged</i>	An integer value that specifies the action that was taken to trigger the event.
<i>NewIndex</i>	The index of the current column after the change has occurred.

### Settings

The settings for *WhatChanged* are:

There are [constants](#) available for the settings of this parameter.

### Remarks

Other events triggered by column movement are triggered before the move occurs.

The value of the column index returned by *NewIndex* will usually be the same as it was before the event occurred. It will only change when moving or swapping columns between groups.

## **AfterPosChanged Event Applies To**

SSDBGrid

## **AfterPosChanged Event See Also**

[ColMove](#) event

[ColSwap](#) event

[GrpMove](#) event

[GrpSwap](#) event

## AfterUpdate Event

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Occurs after the user updates the current row.

### Syntax

**Sub** control\_ **AfterUpdate** (*RtnDispErrMsg* **As Integer**)

The event parameters are:

Parameter	Description
<i>RtnDispErrMsg</i>	An integer expression that indicates if an error message box should be displayed.

### Remarks

You can use this event to perform any processing required to respond to the occurrence of the update. This event also provide a way to trap errors associated with updating the data in the database with data from the control.

For more information on how to handle data-related errors, see "[How the Data Grid handles data validation and error checking.](#)"

## **AfterUpdate Event Applies To**

SSDBGrid

## **AfterUpdate Event See Also**

[AfterColUpdate](#) event

[AfterDelete](#) event

[AfterInsert](#) event

[BeforeColUpdate](#) event

[BeforeDelete](#) event

[BeforeInsert](#) event

[BeforeUpdate](#) event

[How the Data Grid handles data validation and error checking](#)

## Alignment Property

[See Also](#)

[Applies To](#)

### Description

For [SSDBData](#), Determines how the text will be aligned within the caption area.

For [SSDBOptSet](#), Determines how the button will be displayed with respect to the text.

### Syntax

*object* . **Alignment**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the alignment position to take as described in Settings.

### Settings (SSDBData)

<b>Setting</b>	<b>Description</b>
0	Left Justify - Top
1	Left Justify - Middle
2	Left Justify - Bottom
3	Right Justify - Top
4	Right Justify - Middle
5	Right Justify - Bottom
6	Center - Top
7	(Default) Center - Middle
8	Center - Bottom

There are [constants](#) available for the settings of this property.

### Settings (SSDBOptSet)

<b>Setting</b>	<b>Description</b>
0	(Default) Left Justify
1	Right Justify

There are [constants](#) available for the settings of this property.

## Alignment Property (Column Object)

Applies To

### Description

Determines how the text will be aligned within the column.

### Syntax

*object* . **Alignment**[= *number* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the alignment position to take as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
0	Left Justify
1	Right Justify
2	Center Justify

There are constants available for the settings of this property.

### Remarks

The default setting is determined by the control based on the data type.



## **Alignment Property (Column Object) Applies To**

Column Object

## **Alignment Property Applies To**

SSDBData

SSDBOptSet

## **Alignment Property See Also**

CaptionAlignment property

## AlignmentPicture Property

[See Also](#)

[Applies To](#)

### Description

Determines the alignment of the graphic specified in the **Picture** property.

### Syntax

*object* . **AlignmentPicture**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the alignment of the picture as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
0	(Default) Left of text
1	Right of text
2	Fit to Caption
3	Tile

There are [constants](#) available for the settings of this property.

## **AlignmentPicture Property Applies To**

StyleSet Object

**AlignmentPicture Property See Also**

[AlignmentText](#)

## AlignmentText Property

[See Also](#)

[Applies To](#)

### Description

Determines how the text will be aligned.

### Syntax

*object* . **AlignmentText**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the alignment position to take as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
0	Left Justify
1	Right Justify
2	Center Justify

There are [constants](#) available for the settings of this property.

## **AlignmentText Property Applies To**

StyleSet Object



**AlignmentText Property See Also**

[AlignmentPicture](#)

## **AllowAddNew Property Applies To**

SSDBGrid

## AllowAddNew Property

[See Also](#)

[Applies To](#)

### Description

Determines if new records are allowed to be added to the data grid by the user.

### Syntax

*object* . **AllowAddNew**[ = *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether new records can be added to the grid, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	The user can add new records to the grid.
<b>False</b>	(Default) The user can not add new records to the grid.

### Remarks

When enabled, new records can be appended at the bottom of the grid in the row denoted by an asterisk (\*).

## **AllowAddNew Property See Also**

[AllowDelete](#)

[AllowUpdate](#)

## **AllowColumnMoving Property Applies To**

SSDBGrid

## AllowColumnMoving Property

[See Also](#)

[Applies To](#)

### Description

Determines if columns can be moved by the user, and if so, the scope of the move.

### Syntax

*object* . **AllowColumnMoving**[= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the position to take as described in Settings.

### Settings

Setting	Description
0	Column moving is not allowed.
1	(Default) Columns can be moved only within a group.
2	Columns can be moved anywhere including between groups.

There are [constants](#) available for the settings of this property.

### Remarks

Columns can be moved by clicking on the header of the column to move, and dragging it to the new location.

When columns are moved from one group to another, the column is removed from the source group and inserted in the destination group.

Swapping or moving columns does not change the name or number of the column, that is column number 2 is still column number 2 despite being moved.

## **AllowColumnMoving Property See Also**

[AllowColumnSizing](#)

[AllowColumnSwapping](#)

## **AllowColumnShrinking Property Applies To** SSDBGrid



## AllowColumnShrinking Property

[See Also](#)

[Applies To](#)

### Description

Determines if columns can be shrunk to their minimum width by the user clicking the right mouse button on the header.

### Syntax

*object* . **AllowColumnShrinking**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the column can be shrunk to its minimum size, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) The user can shrink the group.
<b>False</b>	The user can not shrink the group.

### Remarks

Shrinking columns is useful to bring scrolled columns into view quickly.

**AllowColumnShrinking Property See Also**

[AllowGroupShrinking](#)

## AllowColumnSizing Property

[See Also](#)

[Applies To](#)

### Description

Determines if columns in the grid can be resized by the user or are fixed in width.

### Syntax

*object* . **AllowColumnSizing**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether columns can be resized, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) Columns can be resized by the user.
<b>False</b>	Columns can not be resized by the user.

### Remarks

Columns can be resized by dragging the right edge of the column left for smaller or right for larger.

## **AllowColumnSizing Property Applies To**

SSDBGrid

## **AllowColumnSizing Property See Also**

[AllowColumnMoving](#)

[AllowColumnSwapping](#)

## AllowColumnSwapping Property

[See Also](#)

[Applies To](#)

### Description

Determines if the position of two columns can be swapped by the user, and if so, the scope of the swap.

### Syntax

*object* . **AllowColumnSwapping**[= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the alignment position as described in Settings.

### Settings

Setting	Description
0	Column swapping is not allowed.
1	(Default) Columns can be swapped only within a group.
2	Columns can be swapped anywhere.

There are [constants](#) available for the settings of this property.

### Remarks

Swapping columns is accomplished by first clicking the header of one of the columns you wish to swap, then clicking on the dropdown list that appears, and finally selecting the name of the column you wish to swap with.

When columns are moved from one group to another, the column is removed from the source group and inserted in the destination group.

Swapping or moving columns does not change the name or number of the column, that is column number 2 is still column number 2 despite being moved.

The following example screens show how the fields "Company Name" and "Address" are swapped:

Form1

*MultiLevel Grid*

*Company Info*

<i>Name</i>	<i>Company Name</i>
<i>Address</i>	
ACM	Association for Computing 11 W. 42nd St., 3rd flr.
rgtrtutty	Addison-Wesley Publishing Co Rte 128

**Before Swap**

Form1

*MultiLevel Grid*

*Company Info*

<i>Name</i>	<i>Address</i>
<i>Company Name</i>	
ACM	11 W. 42nd St., 3rd flr. Association for Computing Machinery
rgtrtutty	Rte 128 Addison-Wesley Publishing Co Inc.

**After Swap**

## **AllowColumnSwapping Property Applies To** SSDBGrid



## **AllowColumnSwapping Property See Also**

[AllowColumnMoving](#)

[AllowColumnSizing](#)

## AllowDelete Property

[See Also](#)

[Applies To](#)

### Description

Determines if rows in the grid can be deleted by the user.

### Syntax

*object* . **AllowDelete**[= *boolean* ]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether rows can be deleted, as described in Settings.

### Settings

Setting	Description
<b>True</b>	Rows can be deleted by the user.
<b>False</b>	(Default) Rows can not be deleted by the user.

### Remarks

Rows can be deleted by selecting the row and then pressing the "Delete" key. Multiple rows can be deleted by selecting them using either the *Ctrl* or *Shift* key and then pressing the *Delete* key (provided *MultiSelect* or *MultiSelectRange* is specified for the **SelectTypeRow** property).

## **AllowDelete Property Applies To**

SSDBGrid

## **AllowDelete Property See Also**

[AllowAddNew](#)

[AllowUpdate](#)

## AllowDragDrop Property

Applies To

### Description

Determines if drag and drop of cell data can be used.

### Syntax

*object* . **AllowDragDrop**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether drag and drop of cell data is available, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) Cell data can be moved by drag and drop.
<b>False</b>	Cell data can not be moved by drag and drop.

### Remarks

Drag and drop of cell data allows you to drag data from a cell to another cell or another application that supports OLE drag and drop.

To copy data without moving it from the source, hold down the *Ctrl* key when selecting.

## **AllowDragDrop Property Applies To**

SSDBGrid

## AllowGroupMoving Property

[See Also](#)

[Applies To](#)

### Description

Determines if groups can be moved by the user.

### Syntax

*object* . **AllowGroupMoving**[= *boolean* ]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether groups can be moved, as described in Settings.

### Settings

Setting	Description
<b>True</b>	(Default) Groups can be moved by the user.
<b>False</b>	Groups can not be moved by the user.

### Remarks

Groups can be moved by clicking on the header of the group to move, and dropping it on the new location.

Swapping or moving groups does not change the name or number of the group, that is group number 2 is still group number 2 despite being moved.

## **AllowGroupMoving Property Applies To**

SSDBGrid



## **AllowGroupMoving Property See Also**

[AllowGroupSizing](#)

[AllowGroupSwapping](#)

## AllowGroupShrinking Property

[See Also](#)

[Applies To](#)

### Description

Determines if groups can be shrunk to their minimum width by the user clicking the right mouse button on the header.

### Syntax

*object* . **AllowGroupShrinking**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the group can be shrunk, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) The user can shrink the group.
<b>False</b>	The user can not shrink the group.

### Remarks

Shrinking groups is useful to bring scrolled groups into view quickly.

## **AllowGroupShrinking Property Applies To**

SSDBGrid

**AllowGroupShrinking Property See Also**

[AllowColumnShrinking](#)

## AllowGroupSizing Property

[See Also](#)

[Applies To](#)

### Description

Determines if groups in the grid can be resized by the user or are fixed-in width.

### Syntax

*object* . **AllowGroupSizing**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether groups can be resized, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) Groups can be resized by the user.
<b>False</b>	Groups can not be resized by the user.

### Remarks

Groups can be resized by dragging the right edge of the group in the direction to resize, either left for smaller or right for larger.

## **AllowGroupSizing Property Applies To**

SSDBGrid

## **AllowGroupSizing Property See Also**

[AllowGroupMoving](#)

[AllowGroupSwapping](#)

## AllowGroupSwapping Property

[See Also](#)

[Applies To](#)

### Description

Determines if the position of two groups can be swapped by the user.

### Syntax

*object* . **AllowGroupSwapping**[= *boolean* ]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether groups can be swapped, as described in Settings.

### Settings

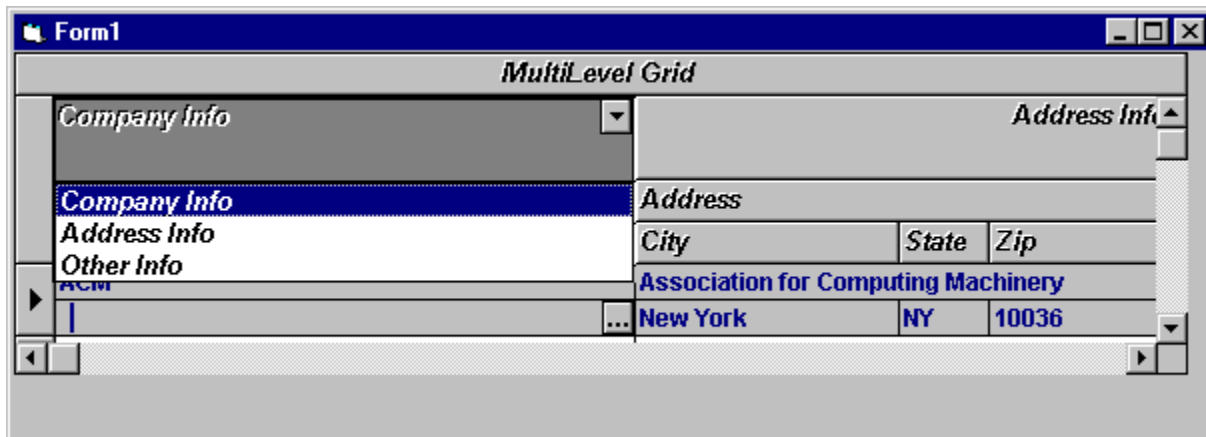
Setting	Description
<b>True</b>	(Default) Groups can be swapped by the user.
<b>False</b>	Groups can not be swapped by the user.

### Remarks

Swapping groups is accomplished by first clicking the header of one of the groups you wish to swap, then clicking on the dropdown list that appears, and finally selecting the name of the group you wish to swap with.

Swapping or moving groups does not change the name or number of the group, that is group number 2 is still group number 2 despite being moved.

The following example screens show how the groups "Company Info" and "Address Info" are swapped:



Before Swap



Form1

*MultiLevel Grid*

<i>Address Info</i>			<i>Company Info</i>
<i>Address</i>			<i>Name</i>
<i>City</i>	<i>State</i>	<i>Zip</i>	<i>Company Name</i>
Association for Computing Machinery			ACM
New York	NY	10036	

After Swap

## **AllowGroupSwapping Property Applies To** SSDBGrid

## **AllowGroupSwapping Property See Also**

[AllowGroupMoving](#)

[AllowGroupSizing](#)

## AllowInput Property

[See Also](#)

[Applies To](#)

### Description

Determines if the user can make changes to data in the control.

### Syntax

*object* . **AllowInput**[ = *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the user can make changes to data in the control, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) The user can make changes to the data.
<b>False</b>	The user can not make changes to the data.

### Remarks

Text can be changed by typing in the edit portion of the control.

## **AllowInput Property Applies To**

SSDBCombo

## **AllowInput Property See Also**

[AllowNull](#)

## AllowNull Property

Applies To

### Description

Determines if the edit portion of the SSDBCombo control permits the entry of a null value.

### Syntax

*object* . **AllowNull**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the entry of a null value is permitted, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) SSDBCombo will allow a null value to be entered into the edit portion.
<b>False</b>	SSDBCombo will not allow a null value to be entered into the edit portion.

### Remarks

If **AllowNull** is set to True and no text is entered in the control, an empty string ("") will be saved to the database.

## **AllowNull Property Applies To**

SSDBCombo



## AllowRowSizing Property

Applies To

### Description

Determines if row heights in the grid can be resized by the user.

### Syntax

*object* . **AllowRowSizing**[= *boolean* ]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether rows can be resized, as described in Settings.

### Settings

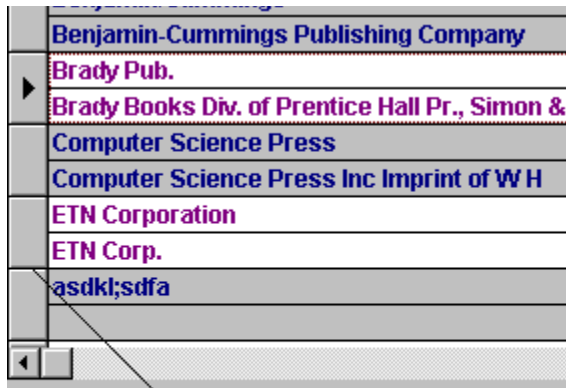
Setting	Description
<b>True</b>	(Default) Row height can be resized by the user.
<b>False</b>	Row height can not be resized by the user.

### Remarks

Changes to row height apply to all rows in the grid.

If **RowHeight** = 0, the grid defaults to a sizing based on the font metrics of the grid text. The **RowHeight** setting can not be smaller than the size of the grid text.

Rows can be resized by dragging the bottom edge of the row's selection column in the direction to resize, either up for smaller or down for larger.



Select and drag  
the bottom edge  
to resize

## **AllowRowSizing Property Applies To**

SSDBGrid

## AllowSizing Property

[See Also](#)

[Applies To](#)

### Description

Determines if the specified object can be resized by the user.

### Syntax

*object* . **AllowSizing**[ = *boolean* ]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the specified object can be resized by the user, as described in Settings.

### Settings

Setting	Description
<b>True</b>	(Default) The object can be resized by the user.
<b>False</b>	The object can not be resized by the user.

### Remarks

Objects can be resized by dragging the right edge of the column in the direction to resize, either left for smaller or right for larger.

This property differs from **AllowColumnSizing** in that **AllowSizing** only affects an individual object, whereas **AllowColumnSizing** is a control-level property.

Individual column settings with **AllowSizing** overrides global settings made with **AllowColumnSizing**.

Groups can be resized by dragging the right edge of the group in the direction to resize, either left for smaller or right for larger.

## **AllowSizing Property Applies To**

Column object

Group object

## **AllowSizing Property See Also**

[AllowColumnSizing](#)

## AllowUpdate Property

[See Also](#)

[Applies To](#)

### Description

Determines if grid data can be modified by the user.

### Syntax

*object* . **AllowUpdate**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether data can be modified in the grid, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) Data can be modified by the user.
<b>False</b>	Data is read-only and can not be modified by the user.

### Remarks

To edit a cell, click on it with the mouse and enter a new value for the cell. Changes will be accepted automatically when you leave the row. To cancel changes while in the cell, press the ESC key.

## **AllowUpdate Property Applies To**

SSDBGrid

## **AllowUpdate Property See Also**

[AllowAddNew](#)

[AllowDelete](#)



## Anatomy of a Data Combo

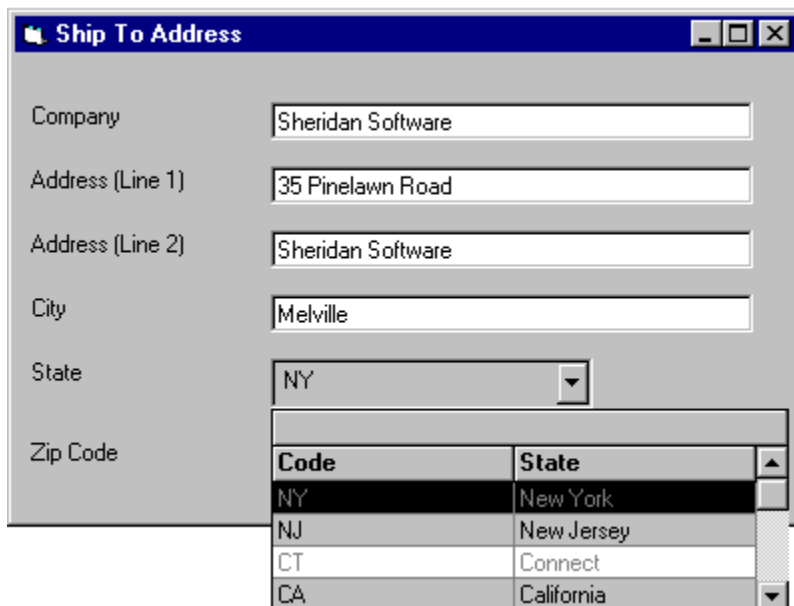
Although there are major fundamental differences, the Data Combo control looks and behaves much like a standard Windows combo box. The major difference is that the Data Combo can be bound to a data control.

The Data Combo is made up of two portions:

- The Edit portion of the Data Combo displays the selected field and allows entry.
- The List portion drops down when the user clicks the dropdown button.

Typically, a combo box is used to allow entry of a particular field while allowing the user to select a value for that field via the dropdown list. With the Data Combo, you can bind the edit portion to a field in one database while the list portion can dropdown a list of values from another.

A classic example is to link the edit portion of the Data Combo to a field in a record set of a data control such as StateCode. Then, link the list portion of the Data Combo to a data control that manages a table having all state codes and translations. The results would look similar to:



The screenshot shows a window titled "Ship To Address" with several text input fields and a dropdown menu. The fields are labeled "Company", "Address (Line 1)", "Address (Line 2)", "City", "State", and "Zip Code". The "State" field is currently set to "NY". Below the "State" field, a dropdown list is open, displaying a table of state codes and names. The table has two columns: "Code" and "State". The rows are: NY (New York), NJ (New Jersey), CT (Connect), and CA (California). The "NY" row is highlighted.

Code	State
NY	New York
NJ	New Jersey
CT	Connect
CA	California

## Anatomy of a Data Grid

Click on an area of the Data Grid to learn more about it:

The screenshot shows a Windows application window titled "Form1" containing a data grid. The grid has a caption "SSDBGrid1.Caption" and is organized into columns: Group #0, Group #1, Column #0 (Name), Column #1 (Company), Column #3 (City), Column #4 (State), Column #5 (Zip), and Column #6 (Paid). A dropdown menu is open over the 'City' column, showing a list of city names. The data rows include various publishers and their details.

Group #0	Group #1	Column #0 (Name)	Column #1 (Company)	Column #3 (City)	Column #4 (State)	Column #5 (Zip)	Column #6 (Paid)
ACM	35 Pinela	Association for Computing	Melville	NY	11747	<input checked="" type="checkbox"/>	
Addison-Wesley	Rte 128	Addison-Wesley Publishing	Reading	MA	01867	<input type="checkbox"/>	
Bantam Books	666 Fifth Ave	Bantam Books Div of: Bantam	Reading	NY	10103	<input checked="" type="checkbox"/>	
Benjamin/Cummings	390 Bridge Pkwy.	Benjamin-Cummings	Redwood City	CA	94065	<input checked="" type="checkbox"/>	
Brady Pub.	15 Columbus Cir.	Brady Books Div. of Prentice	New York	NY	10023	<input type="checkbox"/>	
Computer Science Press	41 Madison Ave	Computer Science Press Inc	New York	NY	10010	<input type="checkbox"/>	
ETN Corporation	Cincinnati	ETN Corp.	Dubuque	PA	17754-9433	<input checked="" type="checkbox"/>	
Gale	Englewood Cliffs	Gale Research, Incorporated	Glenview	MI	48226-4094	<input type="checkbox"/>	
IEEE	Homewood	IEEE Computer Society Press	Lanham	MD	20786-4302	<input checked="" type="checkbox"/>	
Intertext	Menlo Park	Intertext	Anchorage	AK	99508	<input type="checkbox"/>	
M&T Books	2633 E. 17th Ave.	M&T Books Div. of: M&T	Redwood City	CA	94065-4302	<input checked="" type="checkbox"/>	

## Anatomy of the Enhanced Data Control



### First Record

Jumps to the first record in the database. This button is displayed/hidden by the **ShowFirstLastButtons** property.



### Previous Page

Jumps to the previous page in the database. A page is determined by the setting of **PageValue**. This button is displayed/hidden by the **ShowPageButtons** property.



### Previous Record

Jumps to the previous record in the database. This button is displayed/hidden by the **ShowPrevNextButtons** property.



### Add Record

Adds a new record to the end of the database. This button is displayed/hidden by the **ShowAddButton** property.



### Cancel Add

Cancels the adding of a new record to the database. This button is displayed/hidden by the **ShowCancelButton** property.



### Delete Record

Deletes a record from the database. This button is displayed/hidden by the **ShowDeleteButton** property.



### Update Record

Updates the selected record in the database. This button is displayed/hidden by the **ShowUpdateButton** property.



### Add Bookmark

Adds a bookmark for the current record. This button is displayed/hidden by the **ShowBookmarksButton** property.



### Clear All Bookmarks

Clears all stored bookmarks. This button is displayed/hidden by the **ShowBookmarksButton** property.



### Current Record

When the **DataField** property is set, the active record is displayed. When **DataField** is left blank, the **Caption** is displayed.



### **Goto Bookmark**

Presents a list of all stored bookmarks (up to a user-definable limit of 100). This button is displayed/hidden by the **ShowBookmarksButton** property.



### **Find Record**

Invokes the Find dialog, allowing the user to search the database.



### **Find Previous Record**

Searches backwards in the database for the next occurrence of data specified in the Find dialog.



### **Find Next Record**

Searches forwards in the database for the next occurrence of data specified in the Find dialog.



### **Next Record**

Jumps to the next record in the database. This button is displayed/hidden by the **ShowPrevNextButtons** property.



### **Next Page**

Jumps to the next page in the database. A page is determined by the setting of **PageValue**. This button is displayed/hidden by the **ShowPageButtons** property.



### **Last Record**

Jumps to the last record in the database. This button is displayed/hidden by the **ShowFirstLastButtons** property.



## **Applying pictures based on cell contents**

To apply pictures to individual cells, you must first create a StyleSet that contains the picture. The StyleSet can also contain other related attributes (such as font and color) that you wish to apply.

Then you use the RowLoaded event to apply the StyleSet to an individual cell based on its contents.

1. Create a StyleSet object in code. This code creates a new StyleSet named CellStyle:

```
SSDBGrid1.StyleSets.Add "CellStyle"
```

2. Assign the picture you wish to use to the StyleSet. This code uses a picture called TEST.BMP

```
SSDBGrid1.StyleSets("CellStyle").Picture = "TEST.BMP"
```

3. In the **RowLoaded** event, check the value of individual cells based on the column that corresponds to the cell. **RowLoaded** occurs once for each row. To apply the picture to a cell in the first column that contains the string "Test Value" you would use the following code:

```
SSDBGrid1_RowLoaded(ByVal Bookmark As Variant)
If SSDBGrid1.Columns(0).CellText(Bookmark) = "Test Value" Then
    SSDBGrid1.Columns(0).CellStyleSet "CellStyle"
End If
End Sub
```

To see another example of this procedure, [click here](#).

[◀ Back](#)

## Applying pictures to all cells in a column

To apply pictures to individual cells, you must use the **RowLoaded** event. In this event, apply a `StyleSet` to each cell that contains the picture you wish to display. The procedure is similar to the one covered in [Applying pictures based on cell contents](#) but does not require you to check the contents of the cell for a value.

If you want to apply the pictures in response to an event, such as the user clicking on a command button, you must create a flag that determines whether or not the `StyleSet` will be applied, then check the state of that flag in the **RowLoaded** event. The trigger event (i.e. clicking the button) will set the state of the flag, then perform a **Refresh** on the grid.

[Show me how to do this](#)

[◀ Back](#)

## Attaching a DataGrid to a memory array

A DataGrid can retrieve data from and store data to a memory array. You must create an array to hold the data, set up and unbound DataGrid to present the data, then add code to the DataGrid's events to store and retrieve data to and from the array.

1. In code, declare a memory array to hold the data you want to manage. You can create the array globally or in the Form\_Load event of the form that will contain the DataGrid.

You can optionally include code to populate the array with data.

2. Create a DataGrid that corresponds to the dimensions of the array. The DataGrid is best suited to handling data in a two-dimensional array since this corresponds to the rows and columns of the grid. For example, if you have declared an 10 X 50 array, you should create a grid with ten columns. The grid would then have fifty rows.

3. Set the **DataMode** property of the grid to **unbound**.

4. Add code to the **UnboundAddData** event of the grid to store new data in the array.

[Click here](#) to see an example of the code required for this event.

You may want to include code to re-dimension the array "on the fly" as the user adds new data. The example code uses this technique.

5. Add code to the **UnboundReadData** event of the grid to retrieve data from the array as it is needed.

[Click here](#) to see an example of the code required for this event.

6. Add code to the **UnboundDeleteRow** event of the grid to remove data from the array as rows in the grid are deleted. You may also want to add code to the **AfterDelete** event to synchronize the scrollbars with the size of the record set.

[Click here](#) to see an example of the code required for this event.

7. Add code to the **UnboundPositionData** event of the grid to position the grid correctly. This will ensure that data is correctly accessed from the array when the user scrolls through the grid.

[Click here](#) to see an example of the code required for this event.

8. Add code to the **UnboundWriteData** event of the grid to store information modified by the user back into the array.

[Click here](#) to see an example of the code for this event.

## AutoRestore Property

Applies To

### Description

Determines whether text should be restored to previous database value when the ESC key is pressed.

### Syntax

*object* . **AutoRestore**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether text should automatically be restored to its saved value, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) Text will be restored.
<b>False</b>	Text will not be restored.

### Remarks

This property also determines if invalid text will be automatically restored to the last valid text.

In the **TextError** event, the default value of **RtnRestore** is based on this property.

## **AutoRestore Property Applies To**

SSDBCombo



## AutoSize Property

Applies To

### Description

Determines whether the control should automatically be sized to fit the picture specified in the **Picture** property.

### Syntax

*object* . **AutoSize**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the control should be automatically sized, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	Control is sized to the size of the picture assigned in the <b>Picture</b> property.
<b>False</b>	(Default) The control will not automatically resize to a picture.

## **AutoSize Property Applies To**

SSDBCommand

## BIBLIO File Structure

Visual Basic ships with a sample database file called BIBLIO.MDB, which is in Access 2.0 format. Due to the implementation of OCX compatibility by many development tools, it is very possible that users of Data Widgets may not be using Visual Basic as their host development environment.

In light of this fact, the following table describes the structure of the BIBLIO database so that those users can create a database for use with the examples given in this manual.

The BIBLIO database is made up of the following tables:

- Authors
- Publishers
- Title Author
- Titles

### Authors

---

Au_ID	Unique key identifier	Counter (long integer)
Author	Author's name	Text
Year born	Author's birthdate	Integer

### Publishers

---

PubID	Unique key identifier	Counter (long integer)
Name	Short name	Text
Company Name	Full business name	Text
Address	Publisher's address	Text
City	Publisher's city	Text
State	Publisher's state	Text
Zip	Publisher's zip code	Text
Telephone	Publisher's phone number	Text
Fax	Publisher's fax number	Text
Comments	General comments	Memo

### Title Author

---

ISBN	Foreign key into Titles table	Text
Au_ID	Foreign key into Authors table	Long Integer

### Titles

---

Title	Book title	Text
Year Publisher	Publication date	Integer
ISBN	Unique key	Text
PubID	Foreign key into publishers table	Long Integer
Description	Reference info	Text
Notes	General notes	Text
Subject	Keywords	Text
Comments	Description of book contents	Memo

## BackColor Property

Applies To

Example

### Description

For SSDBOptSet, determines the background color for all buttons within the control.

### Syntax

*object* . **BackColor**[ = *color*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>color</i>	A long integer value or constant that determines the background color.

## **BackColor Property Applies To**

Column object

SSDBCombo

SSDBDropDown

SSDBData

SSDBGrid

SSDBOptSet

## BackColorEven Property

[See Also](#)

[Applies To](#)

### Description

Determines the row's background color for even-numbered rows.

### Syntax

*object* . **BackColorEven**[= *color*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>color</i>	A long integer value or constant that determines the color.

## **BackColorEven Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **BackColorEven Property See Also**

[BackColor](#)

[BackColorOdd](#)



## BackColorOdd Property

[See Also](#)

[Applies To](#)

### Description

Determines the row's background color for odd-numbered rows.

### Syntax

*object* . **BackColorOdd**[= *color*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>color</i>	A long integer value or constant that determines the color.

## **BackColorOdd Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **BackColorOdd Property See Also**

[BackColor](#)

[BackColorEven](#)

## BalloonHelp Property

Applies To

### Description

Determines whether balloon help will be displayed.

### Syntax

*object* . **BalloonHelp**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether balloon help will be displayed, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) Balloon Help will be displayed.
<b>False</b>	Balloon Help will not be displayed

### Remarks

In the Enhanced Data Control, Balloon Help is displayed when the mouse is held over a button. Balloon help will display, identifying the button's function.

In the Data Grid, Balloon Help is displayed when the mouse is held over a cell. If the text scrolls past the width, the balloon help will display the entire contents of the cell.

## **BalloonHelp Property Applies To**

SSDBData

SSDBGrid

## BeforeColUpdate Event

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Occurs before data is moved from a cell to the control's copy buffer.

### Syntax

```
Sub control_BeforeColUpdate ([CollIndex As Integer] [OldValue As Variant] [Cancel As Integer])
```

### The event parameters are:

<b>Parameter</b>	<b>Description</b>
<i>CollIndex</i>	An integer expression that specifies the column to be updated.
<i>OldValue</i>	A variant that contains the cell value before the update.
<i>Cancel</i>	An integer expression that specifies whether the operation occurs.

### Remarks

This event is triggered when the user completes editing a row and before the update is written to the control's copy buffer.

By setting *Cancel* = 1, focus from the control can't be moved until the application determines that data can be moved back to the copy buffer.

## **BeforeColUpdate Event Applies To**

SSDBGrid

## **BeforeColUpdate Event See Also**

[AfterColUpdate](#)

[AfterDelete](#)

[AfterInsert](#)

[AfterUpdate](#)

[BeforeDelete](#)

[BeforeInsert](#)

[BeforeUpdate](#)



## BeforeDelete Event

See Also

Applies To

### Description

Occurs after a user attempts to delete a row, but just prior to the row actually being deleted by the control.

### Syntax

**Sub** control\_ **BeforeDelete** (*Cancel* **As Integer**, *DispPromptMsg* **As Integer**)

### The event parameters are:

#### Parameter Description

---

*Cancel* An integer expression that specifies whether the operation occurs.

*DispPromptMsg* An integer expression specifying whether a dialog is displayed asking the user to confirm the deletion.

### Remarks

This event is triggered when the user attempts to delete a row, prior to the actual deletion taking place. Once deleted, the **AfterDelete** event is triggered. The selected row is available in the collection provided by the **SelBookmarks** property.

Setting *Cancel* = 1 causes the deletion to not take place.

## **BeforeDelete Event Applies To**

SSDBGrid

## **BeforeDelete Event See Also**

[AfterColUpdate](#)

[AfterDelete](#)

[AfterInsert](#)

[AfterUpdate](#)

[BeforeColUpdate](#)

[BeforeInsert](#)

[BeforeUpdate](#)

## BeforeInsert Event

[See Also](#)

[Applies To](#)

### Description

Occurs when a user attempts to insert a row, just before the row is actually inserted by the control.

### Syntax

**Sub** control\_ **BeforeInsert** ([*Cancel* **As Integer**])

### The event parameters are:

Parameter	Description
<i>Cancel</i>	An integer expression that specifies whether the operation occurs.

### Remarks

The event is triggered when the user inserts a new row by clicking on the \* next to the last row. The **BeforeUpdate** event immediately follows this event.

Setting *Cancel* = 1 causes the insertion to not take place and clears any data the user entered.

At the time the **BeforeInsert** event is triggered, the data has not been committed to the database, and resides in the copy buffer. Only when the **AfterInsert** event takes place is the data moved from the buffer to the database.

## **BeforeInsert Event Applies To**

SSDBGrid

## **BeforeInsert Event See Also**

[AfterColUpdate](#)

[AfterDelete](#)

[AfterInsert](#)

[AfterUpdate](#)

[BeforeColUpdate](#)

[BeforeDelete](#)

[BeforeUpdate](#)

## BeforeRowColChange Event

See Also

Applies To

### Description

Occurs before the user changes the current row or column.

### Syntax

**Sub** control\_ **BeforeRowColChange** (*Cancel* **As Integer**)

### The event parameters are:

<b>Parameter</b>	<b>Description</b>
------------------	--------------------

---

<i>Cancel</i>	An integer expression specifying whether the row and/or column change will occur.
---------------	---

### Remarks

This event is triggered just after the current column or row changes. If both the column and row change, the event will only be fired once. Unless canceled, the **RowColChange** event is triggered once the focus has moved to the new cell.

Setting *Cancel* = 1 prevents the focus from changing, and halts execution of the **RowColChange** event.

## **BeforeRowColChange Event Applies To**

SSDBGrid



## **BeforeRowColChange Event See Also**

[RowColChange](#)

[RowLoaded](#)

## BeforeUpdate Event

[See Also](#)

[Applies To](#)

### Description

Occurs before changes a user has made are committed to the database.

### Syntax

**Sub** control\_ **BeforeUpdate** ([*Cancel* **As Integer**])

### The event parameters are:

<b>Parameter</b>	<b>Description</b>
------------------	--------------------

---

<i>Cancel</i>	An integer expression that specifies whether the operation occurs.
---------------	--

### Remarks

When the user moves to another row, data is moved from the control's copy buffer to the database. Prior to moving the data from the copy buffer to the database, this event is triggered. Unless canceled, the **AfterUpdate** event is triggered once the data has been written to the database.

Setting *Cancel* = 1 causes the update to not be written to the database, and halts execution of the **AfterUpdate** event.

## **BeforeUpdate Event Applies To**

SSDBGrid

## **BeforeUpdate Event See Also**

[AfterColUpdate](#)

[AfterDelete](#)

[AfterInsert](#)

[AfterUpdate](#)

[BeforeColUpdate](#)

[BeforeDelete](#)

[BeforeInsert](#)

## BevelColorFace, BevelColorFrame, BevelColorHighlight, BevelColorShadow Properties

Applies To

### Description

Determines the colors used to draw respective parts of the control.

### Syntax

*object* . **BevelColorFace**[= *color*]

*object* . **BevelColorFrame**[= *color*]

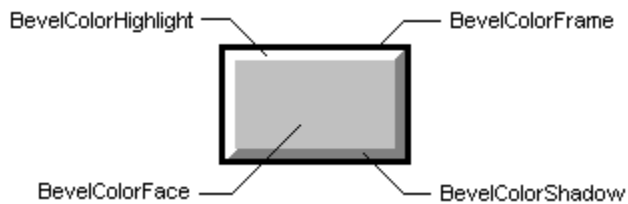
*object* . **BevelColorHighlight**[= *color*]

*object* . **BevelColorShadow**[= *color*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>color</i>	A long integer value or constant that determines the 3D effect color.

### Remarks

When the control needs to draw a bevel, it uses the colors specified in these properties. These properties refer to the different parts of the bevels as shown in the diagram.



## **BevelColorFace, BevelColorFrame, BevelColorHighlight, BevelColorShadow Properties Applies To**

SSDBCombo

SSDBCommand

SSDBData

SSDBDropDown

SSDBGrid

SSDBOptSet

## BevelColorScheme Property

[See Also](#)

[Applies To](#)

### Description

Determines the color scheme for the parts of the control.

### Syntax

*object* . **BevelColorScheme**[= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the color scheme for the 3D bevels, as described in Settings.

### Settings

Setting	Description
0	Gray Colors - uses white, black, light gray and dark gray from the standard Windows VGA palette.
1	System Colors - uses the system color values specified in the Windows Control Panel.
2	(Default) Custom Colors - uses the color values specified in the BevelColorFace, BevelColorHighlight, BevelColorFrame and BevelColorShadow properties.

There are [constants](#) available for the settings of this property.

### Remarks

This is how the preset color schemes are applied:

#### Gray Colors:

Frame = Black  
Face = Light Gray  
Shadow = Dark Gray  
Highlight = White

#### System Colors:

Frame = Window Frame  
Face = Button Face  
Shadow = Button Shadow  
Highlight = Button Highlight

#### Custom Colors:

Uses the colors set in the BevelColor properties.





## **BevelColorScheme Property Applies To**

SSDBCombo

SSDBCommand

SSDBData

SSDBDropDown

SSDBGrid

SSDBOptSet

## **BevelColorScheme Property See Also**

[BevelColorFace](#)

[BevelColorFrame](#)

[BevelColorHighlight](#)

[BevelColorShadow](#)

## BevelInner Property

[See Also](#)

[Applies To](#)

### Description

Determines the type of inside beveling for the control.

### Syntax

*object* . **BevelInner**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the type of inner bevel to use as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
0	(Default) None. No inner bevel is drawn.
1	Inset. The inner bevel appears as if it is inset into the screen.
2	Raised. The inner bevel appears as if it is raised from the screen.

There are [constants](#) available for the settings of this property.

### Remarks

This property is Windows 95-sensitive.

## **BevellInner Property Applies To**

SSDBData

## **BevelInner Property See Also**

[BevelColorFace](#)

[BevelColorFrame](#)

[BevelColorHighlight](#)

[BevelColorScheme](#)

[BevelColorShadow](#)

## BevelOuter Property

[See Also](#)

[Applies To](#)

### Description

Determines the type of outside beveling for the control.

### Syntax

*object* . **BevelOuter**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the type of inner bevel to use as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
0	None. No outer bevel is drawn.
1	Inset. The outer bevel appears as if it is inset into the screen.
2	(Default) Raised. The outer bevel appears as if it is raised from the screen.

There are [constants](#) available for the settings of this property.

## **BevelOuter Property Applies To**

SSDBData

## **BevelOuter Property See Also**

[BevelInner](#)

[BevelColorHighlight](#)

[BevelColorScheme](#)

[BevelColorShadow](#)



## BevelType Property

Applies To

### Description

Sets the type of bevel to be used around the control.

### Syntax

*object* . **BevelType**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the type of bevel to use, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
0	None
1	(Default) Inset
2	Raised

There are constants available for the settings of this property.

## **BevelType Property Applies To**

SSDBCombo

SSDBDropDown

## BevelWidth Property

[See Also](#)

[Applies To](#)

### Description

Determines the width of bevels which determines the amount of the 3D shadow effect.

### Syntax

*object* . **BevelWidth**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the width of the bevel, which determines the amount of the 3D shadow effect.

### Remarks

Valid range is 0 to 10. The default value is 1.

## **BevelWidth Property Applies To**

SSDBCombo

SSDBCommand

SSDBData

## **BevelWidth Property See Also**

[BevelInner](#)

[BevelOuter](#)

## Binding a Data Command to a Data Control Across Forms

Starting with Visual Basic 4.0, binding to a data control across forms is a relatively easy task. In the past, the only way to accomplish this task was to set a **DataSourceHwnd** property to point to the **hWnd** of a data control. You are now able to set the data controls to look at one another.

To bind o a data control across forms:

1. On Form1, create a standard Data Control (Data1), setting the Database and RecordSource properties to point to the database table you want to use.
2. On Form2, create a standard Data Control (Data1), setting the Database and RecordSource properties to point to the same database table as Step 1.
3. In the Form\_Load() section of Form2, add the code `Set Data1.Recordset = Form1!Data1.Recordset`.
4. On Form 2, create a Data Command button, setting the DataSource property to Data1.

The data controls are now bound across forms, that is, actions to the data control on either form are automatically reflected by one another.

## Binding a Data DropDown to a Data Control Across Forms

Starting with Visual Basic 4.0, binding to a data control across forms is a relatively easy task. In the past, the only way to accomplish this task was to set a **DataSourceHwnd** property to point to the **hWnd** of a data control. You are now able to set the data controls to look at one another.

To bind to a data control across forms:

1. On Form1, create a standard Data Control (Data1), setting the Database and RecordSource properties to point to the database table you want to use.
2. On Form2, create a standard Data Control (Data1), setting the Database and RecordSource properties to point to the same database table as Step 1.
3. In the Form\_Load() section of Form2, add the code `Set Data1.Recordset = Form1!Data1.Recordset`.
4. On Form 2, create a Data DropDown, setting the DataSource property to Data1.

The data controls are now bound across forms, that is, actions to the data control on either form are automatically reflected by one another.

## Binding a DataSet to a Data Control Across Forms

Starting with Visual Basic 4.0, binding to a data control across forms is a relatively easy task. In the past, the only way to accomplish this task was to set a **DataSourceHwnd** property to point to the **hWnd** of a data control. You are now able to set the data controls to look at one another.

To bind to a data control across forms:

1. On Form1, create a standard Data Control (Data1), setting the Database and RecordSource properties to point to the database table you want to use.
2. On Form2, create a standard Data Control (Data1), setting the Database and RecordSource properties to point to the same database table as Step 1.
3. In the Form\_Load() section of Form2, add the code `Set Data1.Recordset = Form1!Data1.Recordset`.
4. On Form 2, create an Enhanced Data Control, setting the DataSource property to Data1.

The data controls are now bound across forms, that is, actions to the data control on either form are automatically reflected by one another.



## Binding an EDC to a Data Control Across Forms

Starting with Visual Basic 4.0, binding to a data control across forms is a relatively easy task. In the past, the only way to accomplish this task was to set a **DataSourceHwnd** property to point to the **hWnd** of a data control. You are now able to set the data controls to look at one another.

To bind to a data control across forms:

1. On Form1, create a standard Data Control (Data1), setting the Database and RecordSource properties to point to the database table you want to use.
2. On Form2, create a standard Data Control (Data1), setting the Database and RecordSource properties to point to the same database table as Step 1.
3. In the Form\_Load() section of Form2, add the code `Set Data1.Recordset = Form1!Data1.Recordset`.
4. On Form 2, create an Enhanced Data Control, setting the DataSource property to Data1.

The data controls are now bound across forms, that is, actions to the data control on either form are automatically reflected by one another.

## Binding the Data Combo to a Data Control Across Forms

Starting with Visual Basic 4.0, binding to a data control across forms is a relatively easy task. In the past, the only way to accomplish this task was to set a **DataSourceHwnd** property to point to the **hWnd** of a data control. You are now able to set the data controls to look at one another. This functionality is useful for the edit portion of the Data Combo.

To bind to a data control across forms:

1. On Form1, create a standard Data Control (Data1), setting the Database and RecordSource properties to point to the database table you want to use.
2. On Form2, create a standard Data Control (Data1), setting the Database and RecordSource properties to point to the same database table as Step 1.
3. In the Form\_Load() section of Form2, add the code `Set Data1.Recordset = Form1!Data1.Recordset`.
4. On Form 2, create a Data Combo, setting the DataSource property to Data1.

The data controls are now bound across forms, that is, actions to the data control on either form are automatically reflected by one another.

## Binding to a Data Control Across Forms

Starting with Visual Basic 4.0, binding to a data control across forms is a relatively easy task. In the past, the only way to accomplish this task was to set a **DataSourceHwnd** property to point to the **hWnd** of a data control. You are now able to set the data controls to look at one another.

To bind to a data control across forms:

1. On Form1, create a standard Data Control (Data1), setting the Database and RecordSource properties to point to the database table you want to use.
2. On Form2, create a standard Data Control (Data1), setting the Database and RecordSource properties to point to the same database table as Step 1.
3. In the Form\_Load() section of Form2, add the code `Set Data1.Recordset = Form1!Data1.Recordset`.
4. On Form 2, create a Data Grid Control, setting the DataSource property to Data1.

The data controls are now bound across forms, that is, actions to the data control on either form are automatically reflected by one another.

## Bold Property

Applies To

Example

### Description

Returns or sets the font style of the specified **Font** or **Headfont** object to either bold or non-bold.

### Syntax

*object* . **Bold**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying the font style, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	Turns on bold formatting.
<b>False</b>	(Default) Turns off bold formatting.

### Remarks

The **Font** and **Headfont** objects are not directly available at design time. At design time, you set the **Bold** property through the control's **Font** or **Headfont** property. At runtime, you can set **Bold** directly by specifying its settings for the appropriate **Font/Headfont** object.

## **Bold Property Applies To**

Font object

Headfont object

## Bookmark Object

Applies To

### Description

A bookmark object contains information that uniquely specifies a record in the database. It is used to remember positions of individual records and to return the recordset to those positions.

### Properties

---

String

Value

### Remarks

Bookmark objects are used to populate the dropdown list of marked records in the Enhanced Data Control.

The Bookmark object provides a way for the programmer to easily examine and act on the database bookmarks stored in the Enhanced Data Control's dropdown list.

Each bookmark contains a value and a string. Also see the note on the **Bookmark** property.

## **Bookmark Object Applies To**

Bookmarks collection

## Bookmark Property (ssRowBuffer only)

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Sets or returns the bookmark value of the selected object.

### Syntax

*object* . **Bookmark**(*index* ) [= *variant* ]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>index</i>	The row in the ssRowBuffer object to which the bookmark refers.
<i>variant</i>	A variant expression specifying the bookmark value.

### Remarks

The **Bookmark** property of the ssRowBuffer object is similar in function to the standard **Bookmark** property, although its implementation is slightly different. Because you supply the bookmark to the row buffer through code, it can be any type of value. You are responsible for ensuring the uniqueness and consistency of your bookmarks.

The **Bookmark** property of the ssRowBuffer is a property array. The row in the row buffer to which the bookmark corresponds is determined by the *index*, which is always a value from 0 to 9.

Also see the [note on the \*\*Bookmark\*\* property](#).



## **BookmarksToKeep Property Applies To**

ssRowBuffer object

## **Bookmark Property See Also**

ReadType property

ssRowBuffer object

UnboundReadData event

## BookmarkDisplay Property

[See Also](#)

[Applies To](#)

### Description

Determines the method in which bookmarks are displayed in the dropdown bookmark list.

### Syntax

*object* . **BookmarkDisplay**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the bookmark display method as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
0	(Default) Displays bookmarks in historical order.
1	Displays bookmarks in most recently added order.
2	Displays bookmarks in sorted order, sorted by string property.

There are [constants](#) available for the settings of this property.

## **BookmarkDisplay Property Applies To**

SSDBData

**BookmarkDisplay Property See Also**

[BookmarksToKeep](#)

## Bookmarks Collection

[See Also](#)

[Applies To](#)

### Description

The bookmark collection represents a group of bookmark objects.

### Properties

---

[Count](#)

[Item](#)

### Methods

---

[Add](#)

[Remove](#)

[RemoveAll](#)

### Remarks

There can be from 0 to 99 Bookmark objects in this collection. There is only one Bookmarks collection per Enhanced Data Control.

## **Bookmarks Collection Applies To**

SSDBData

## **Bookmarks Collection See Also**

[Bookmark](#) Object



## BookmarksToKeep Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the maximum number of bookmarks to keep.

### Syntax

*object* . **BookmarksToKeep**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the maximum number of bookmarks to keep.

### Remarks

The valid range for this property is 1-100 with a default value of 10.

## **BookmarksToKeep Property Applies To**

SSDBData

## **BookmarksToKeep Property See Also**

[BookmarkDisplay](#)

## BorderStyle Property

Applies To

### Description

Sets or returns the border style of the control.

### Syntax

*object* . **BorderStyle**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the border style as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
0	No border will be displayed.
1	(Default) A fixed single border will be displayed.

There are constants available for the settings of this property.

### Remarks

For **SSDBCombo** and **SSBBDropDown**, this property affects the dropdown portion only.

## **BorderStyle Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## BorderWidth Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the width of the space between the outer and inner bevels.

### Syntax

*object* . **BorderWidth**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the width of the control's border.

### Remarks

The valid range for this property is 0 to 10 with a default value of 3.

## **BorderWidth Property Applies To**

SSDBData

## **BorderWidth Property See Also**

[BevelInner](#)

[BevelOuter](#)

[BevelColorFace](#)



## **BtnClick Event**

Applies To

### **Description**

Fired when a user clicks on a button within a cell.

### **Syntax**

**Sub** control\_ **BtnClick** ()

### **Remarks**

This event will only fire when the column's **Style** property is set to 1 (Edit Button) or 4 (Button) and the button is clicked.

## **BtnClick Event Applies To**

SSDBGrid

## Button Object

Applies To

### Description

The button object represents a button in the DataOptionSet.

### Properties

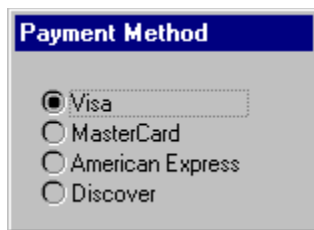
---

<u>Caption</u>	<u>Picture</u>	<u>Value</u>
<u>ColOffset</u>	<u>PictureMetaHeight</u>	<u>Visible</u>
<u>Enabled</u>	<u>PictureMetaWidth</u>	
<u>OptionValue</u>	<u>RowOffset</u>	

### Remarks

Within each button collection, there can be 1 to 100 buttons.

An example of a DataOptionSet with four buttons:



The image shows a dialog box titled "Payment Method" with a blue header. Inside the dialog, there are four radio button options: "Visa" (which is selected), "MasterCard", "American Express", and "Discover".

## **Button Object Applies To**

Buttons collection

## ButtonEnabled Property

[Applies To](#)

[Example](#)

### Description

Determines if an option button can respond to user-generated events.

### Syntax

*object* . **ButtonEnabled**[ = *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the option button can respond to user-generated events, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) The option button will be enabled.
<b>False</b>	The option button will not be enabled

### Remarks

This property allows the option buttons to be enabled or disabled at runtime. For example, you can disable option buttons that don't apply to the current state of the application.

At runtime, you can use a shorthand form to refer to a button, such as:

```
SSDBOptSet1.Buttons(0).Enabled = True
```

## **ButtonEnabled Property Applies To**

SSDBOptSet

## ButtonFromCaption Method

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Returns a button with the specified caption.

### Syntax

*object* . **ButtonFromCaption**(*Caption* **As String**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Caption</i>	The string expression specifying the button caption to search for.

## **ButtonFromCaption Method Applies To**

SSDBOptSet



## **ButtonFromCaption Method See Also**

[ButtonFromPos](#)

## ButtonFromPos Method

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Returns the button which resides at a particular location.

### Syntax

*object* . **ButtonFromPos**(*X* **As Single**, *Y* **As Single**, [*Scale* **As Variant**])

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>X</i>	Determines the X coordinate.
<i>Y</i>	Determines the Y coordinate.
<i>scale</i>	(Optional) Determines the scale that will be used, as described in Settings.

### Settings

The settings for *scale* are:

<b>Setting</b>	<b>Description</b>
0	Twips (Default)
1	Pixels
2	Container Coordinates
3	HiMetric

There are [constants](#) available for the settings of this parameter.

## **ButtonFromPos Method Applies To**

SSDBOptSet

## **ButtonFromPos Method See Also**

[ButtonFromCaption](#) method

[WhereIs](#) method

## ButtonSize Property

Applies To

### Description

Sets or returns the size of each button.

### Syntax

*object* . **ButtonSize**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the size of each button.

### Remarks

If the control is displayed horizontally, this property determines the button width. If the control is displayed vertically, this property determines the button height.

The valid range for this property is 5-100 with a default value of 19.

## **ButtonSize Property Applies To**

SSDBData

## ButtonVisible Property

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Determines whether the selected option button is visible or hidden.

### Syntax

*object* . **ButtonVisible**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the object is visible or hidden, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) Object is visible.
<b>False</b>	Object is hidden.

### Remarks

This is a button-specific property. The button affected by this property is the currently selected button, as determined by the **IndexSelected** property.

## **ButtonVisible Property Applies To**

SSDBOptSet



## **ButtonVisible Property See Also**

[IndexSelected](#)

[Visible](#)

## Buttons Collection

[See Also](#)

[Applies To](#)

[Example](#)

### Description

The button collection represents a group of button objects that you can place on your form.

### Properties

---

[Count](#)

[Item](#)

### Methods

---

[Add](#)

[Remove](#)

[RemoveAll](#)

## **Buttons Collection Applies To**

SSDBOptSet

## **Buttons Collection See Also**

Button Object

## ButtonsAlways Property

[See Also](#)

[Applies To](#)

### Description

Determines whether cells with a button style should be shown at all times, or only when selected.

### Syntax

*object* . **ButtonsAlways**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying the display status of buttons, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	Buttons will be shown at all times for cells that have a button style property.
<b>False</b>	(Default) Buttons will only be displayed when the cell is selected.

### Remarks

This property only applies to columns with a **Style** property of 1 or 3.

## **ButtonsAlways Property Applies To**

Column object

**ButtonsAlways Property See Also**

[Style](#)

## Caption Property

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Determines the caption for the selected object/control.

### Syntax

*object* . **Caption**[= *text*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>text</i>	A string expression that evaluates to the text displayed as the caption.

### Remarks

The appearance of the caption is based on the **Font** and **Font3D** properties. In the case of the **Column** and **Group** objects, the **HeadFont** property determines font settings.



## **Caption Property Applies To**

Button object

Column object

Group object

SSDBCommand

SSDBData

SSDBOptSet

## **Caption Property See Also**

[CaptionAlignment](#)

[Picture](#)

[PictureAlignment](#)

## CaptionAlignment Property

[See Also](#)

[Applies To](#)

### Description

For the **SSDBData** and **SSDBOptSet** controls, determines how the caption will be aligned on each button.

For all others, determines how the caption will be aligned on the object/control.

### Syntax

*object* . **CaptionAlignment**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying which display state to use, as described in Settings.

### Settings (SSDBData, SSDBGrid, SSDBDropDown, Group Object)

<b>Setting</b>	<b>Description</b>
0	Left Justify
1	Right Justify
2	(Default) Center

There are [constants](#) available for the settings of this property.

### Settings (Column Object)

<b>Setting</b>	<b>Description</b>
0	(Default) Left Justify
1	Right Justify
2	Center
3	Follow the alignment specified for the cells.

There are [constants](#) available for the settings of this property.

### Settings (SSDBOptSet)

<b>Setting</b>	<b>Description</b>
0	Right justify
1	(Default) Left justify

There are [constants](#) available for the settings of this property.

### Settings (SSDBCommand)

<b>Setting</b>	<b>Description</b>
0	Left Justify - Top
1	Left Justify - Middle
2	Left Justify - Bottom
3	Right Justify - Top
4	Right Justify - Middle
5	Right Justify - Bottom
6	Center - Top
7	Center - Middle
8	(Default) Center - Bottom

There are constants available for the settings of this property.

## **CaptionAlignment Property Applies To**

Column object

Group object

SSDBData

SSDBGrid

SSDBOptSet

**CaptionAlignment Property See Also**

[PictureAlignment](#)

## Case Property

Applies To

### Description

Sets or returns the case to use for column text.

### Syntax

*object* . **Case**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying which case to use, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
0	(Default) Unchanged
1	lowercase
2	UPPERCASE

There are constants availbel for the settings of this property.

## **Case Property Applies To**

Column object



## CellNavigation Property

[See Also](#)

[Applies To](#)

### Description

Determines how the grid responds to the arrow keys being used when first entering the cell.

### Syntax

*object* . **CellNavigation**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying how arrow keys work when first entering a cell, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
0	(Default) Arrow keys change cells on entry.
1	Arrow keys move within cell on entry.

There are [constants](#) available for the settings of this property.

### Remarks

This property only affects the usage of the arrow keys prior to entering edit mode. Once you begin to edit the value of a cell, the arrow keys automatically operate only within the cell.

## **CellNavigation Property Applies To**

SSDBGrid

**CellNavigation Property See Also**

[RowNavigation](#)

## CellStyleSet Method

[See Also](#)

[Applies To](#)

### Description

Sets the StyleSet for the specified cell.

### Syntax

*object* . **CellStyleSet**(*StyleSet* **As String**, [*RowNum* **As Variant**])

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>StyleSet</i>	A string expression specifying the name of the StyleSet to use.
<i>RowNum</i>	A variant expression specifying the display row number to apply the StyleSet to.

### Remarks

As scrolling occurs, the StyleSet will scroll with the cell until the cell moves out of the display area.

Within the **RowLoaded** event, the row number is ignored because the StyleSet can only be applied to the row being loaded.

Note that the value passed to *RowNum* must be the number of a row in the grid (integer or long) not a bookmark.

## **CellStyleSet Method Applies To**

Column Object

## **CellStyleSet Method See Also**

[ActiveRowStyleSet](#)

[StyleSet](#)

[StyleSets](#) collection

## CellText Method

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Returns the text for a specified row in the grid.

### Syntax

*object* . **CellText**(*Bookmark* **As Variant**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Bookmark</i>	A variant expression evaluating to a row in the grid.

### Remarks

This method returns the value of the cell specified at the row represented by the bookmark as a string.

## **CellText Method Applies To**

Column Object



## **CellText Method See Also**

CellValue method

## CellValue Method

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Returns the underlying data for the specified cell.

### Syntax

*object* . **CellValue**(*Bookmark* **As Variant**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Bookmark</i>	A variant expression evaluating to a row in the grid.

### Remarks

This property is read-only at run time.

This method returns the value of the cell as a variant from the row indicated by the bookmark.

## **CellValue Method Applies To**

Column object

## **CellValue Method See Also**

CellText method

## Change Event

See Also

Applies To

### Description

Occurs when any data within the control is changed by the user.

### Syntax

**Sub** control\_Change ()

### Remarks

In the case of SSDBGrid, this refers to cell data. In the case of SSDBCombo, it refers to the edit portion.

## **Change Event Applies To**

SSDBCombo

SSDBGrid

## **Change Event See Also**

For SSDBGrid

[AfterUpdate](#)

[BeforeColUpdate](#)

[BeforeUpdate](#)

[BtnClick](#)

## **CheckBox Column Style**

The CheckBox column style can be set using the **Style** property



## CheckBox3D Property

Applies To

### Description

Determines whether check boxes on the grid should be displayed with 3D appearance.

### Syntax

*object* . **CheckBox3D**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether checkboxes should be displayed with 3D appearance, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) Checkboxes are displayed with 3D appearance.
<b>False</b>	Checkboxes are displayed with 2D appearance.

## **CheckBox3D Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

[◀ Back](#)

## Clearing formatting and selection information from a grid

There are two methods you can use to reset the attributes of a DataGrid, DataCombo, or DataDropDown.

The **Refresh** method will retrieve a fresh copy of the data underlying the control, effectively removing any data selection attributes, such as selected rows or bookmarks.

The **Reset** method will completely clear any layout or formatting information from the grid. This includes removing any columns that have been created. Any application of StyleSets to grid objects is also destroyed, although the StyleSets themselves remain and can be re-applied.

To completely return a grid to its initial state, use these two methods in conjunction:

```
SSDBGrid1.Refresh  
SSDBGrid1.Reset
```

After executing this code, you would need to recreate the grid layout by adding columns through code.

## Click Event

Applies To

### Description

Occurs when the user clicks the left mouse button over any part of the control.

### Syntax

**Sub** control\_Click (*nPosition* **As Integer**)

The event parameters are:

#### Parameter Description

---

*nPosition* Integer indicating the area of the control being pointed to.

### Remarks

The following values for *nPosition* apply to the Enhanced Data Control

<b>Integer</b>	<b>Area being pointed to</b>
1	Caption Area
2	Bevel Area
3	"First" Button
4	"Last" Button
5	"Previous Page" Button
6	"Next Page" Button
7	"Previous Record" Button
8	"Next Record" Button
9	"Add" Button
10	"Cancel" Button
11	"Update" Button
12	"Delete" Button
13	"Find Next" Button
14	"Find Previous" Button
15	"Find" Button
16	"Add Bookmark" Button
17	"Clear Bookmark" Button
18	"Goto Bookmark" Button

There are constants available for the settings of this parameter.

## Click Event Applies To

SSDBData

## CloseBookmarkDropDown Event

[See Also](#)

[Applies To](#)

### Description

Occurs immediately before the dropdown bookmark list is closed.

### Syntax

**Sub** control\_ **CloseBookmarkDropDown**(*vBookmark* **As Variant**)

<b>Part</b>	<b>Description</b>
<i>vBookmarks</i>	A variant containing the value of the selected bookmark. If none were selected, it will be empty.

## **CloseBookmarkDropDown Event Applies To**

SSDBData

## **CloseBookmarkDropDown Event See Also**

[ShowBookmarkDropDown](#) event



## CloseFindDialog Event

Applies To

Example

### Description

Occurs when the Find dialog is told to close, immediately prior to closing.

### Syntax

**Sub** control\_ **CloseFindDialog**(*FindString* **as Variant**, *Criteria* **As Variant**, *Direction* **As Variant**, *ColToSearch* **As Variant**, *Cancel* **As Integer**)

The event parameters are:

#### Parameter Description

---

<i>FindString</i>	A Variant expression specifying the string to find.
<i>Criteria</i>	A Variant expression specifying the criteria to search for.
<i>ColToSearch</i>	A Variant expression specifying which column in the database to search.
<i>Cancel</i>	An integer expression that specifies whether the operation occurs.

### Settings

The settings for *Criteria* are:

#### Setting Description

---

1	Less Than
2	Less Than or Equal To
3	Equal To
4	Greater Than
5	Greater Than or Equal To
6	Partial Match

There are constants available for the settings of this parameter.

The settings for *Direction* are:

#### Setting Description

---

1	Down (Next)
2	Up (Previous)

There are constants available for the settings of this parameter.

### Remarks

This event is provided so that you may implement your own lookup routines if necessary. By setting *Cancel* to True, you can interrupt the default search action and implement your own algorithm.

The parameters passed to the **CloseFindDialog** event are equivalent to those used with the **Find** method.

## **CloseFindDialog Event Applies To**

SSDBData

## CloseUp Event

[See Also](#)

[Applies To](#)

### Description

Occurs when a dropdown closes up.

### Syntax

```
Sub control_ComboCloseUp ()
```

## **CloseUp Event Applies To**

SSDBCombo

SSBBDropDown

## **CloseUp Event See Also**

ComboDropDown event

## Col Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the current column.

### Syntax

*object* . **Col**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the current column.

### Remarks

Valid range is from 0 to the maximum number of columns created.

## **Col Property Applies To**

SSDBGrid

## **Col Property See Also**

Grp

Row



## ColChanged Property

Applies To

Example

### Description

Returns whether the field in that column for the current row has been modified.

### Syntax

*object* . **ColChanged**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the field has been modified, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) The field has been modified.
<b>False</b>	The field has not been modified.

### Remarks

This property is read-only.

## **ColChanged Property Applies To**

Column object

## ColContaining Method

Applies To

### Description

Returns the index of the column under an x-coordinate.

### Syntax

*object* . **ColContaining**(*X* **As Single**, [*Y* **As Variant**])

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>X</i>	Numeric expression specifying a horizontal coordinate.
<i>Y</i>	Optional. Numeric expression specifying a vertical coordinate for grids with groups.

### Remarks

If the specified coordinate is out of range, an error occurs.

## **ColContaining Method Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## ColMove Event

[See Also](#)

[Applies To](#)

### Description

Occurs before a column is moved.

### Syntax

**Sub** control\_ **ColMove** (*ColIndex* **As Integer**, *NewPos* **As Integer**, *Cancel* **As Integer**)

The event parameters are:

### Parameter Description

---

<i>ColIndex</i>	The column number being moved.
<i>NewPos</i>	An integer expression that specifies the visual position the column is being moved to.
<i>Cancel</i>	An integer expression that specifies whether the operation occurs.

### Remarks

The **ColMove** event is fired after a user moves a column, but before the move is redrawn. You can cancel this event from occurring by setting *Cancel* to true.

Swapping or moving columns does not change the name or number of the column, that is column number 2 is still column number 2 despite being moved.

## **ColMove Event Applies To**

SSDBGrid

## **ColMove Event See Also**

AfterPosChanged event

ColSwap event

GrpMove event

GrpSwap event

## ColOffset Property

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Determines the horizontal offset used to draw the button.

### Syntax

*object* . **ColOffset**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the column offset.

### Remarks

The value range for this property is -32767 to 32767 with a default value of 0.



## **ColOffset Property Applies To**

Button object

SSDBOptSet

**ColOffset Property See Also**

[RowOffset](#)

## ColPosition Method

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Returns the index of the column relative to the collection.

### Syntax

*object* . **ColPosition**(*ColPos* **As Integer**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>ColPos</i>	An integer expression specifying the column as it appears visually.

### Remarks

Remember that columns can be moved, swapped, or made invisible, so the order they appear in is not always their order in the collection.

## **ColPosition Method Applies To**

Group object

SSDBGrid

## **ColPosition Method See Also**

Position

GrpPosition method

## ColResize Event

[See Also](#)

[Applies To](#)

### Description

Occurs before a column is resized.

### Syntax

```
Sub control_ColResize ([ColIndex As Integer] [Cancel As Integer])
```

The event parameters are:

### Parameter Description

---

<i>ColIndex</i>	An integer expression specifying the column being resized.
<i>Cancel</i>	An integer expression that specifies whether the operation occurs.

### Remarks

When the user resizes a column, this event is triggered prior to the column being redrawn. By setting *Cancel* = 1, the original column width is restored and the redraw does not occur.

## **ColResize Event Applies To**

SSDBGrid

## **ColResize Event See Also**

[GrpResize](#)

[RowResize](#)

[SplitterMove](#)



## ColSwap Event

See Also

Applies To

### Description

Occurs before a column is swapped.

### Syntax

**Sub** control\_**ColSwap** (*ColIndex* **As Integer**, *NewPos* **As Integer**, *Cancel* **As Integer**)

The event parameters are:

### Parameter Description

---

<i>ColIndex</i>	The group number being moved.
<i>NewPos</i>	An integer expression that specifies the visual position the column is being swapped to.
<i>Cancel</i>	An integer expression that specifies whether the operation occurs.

### Remarks

The **ColSwap** event is fired after a user swaps a column, but before the swap is redrawn. You can cancel this event from occurring by setting *Cancel* to true.

Swapping or moving columns does not change the name or number of the column, that is column number 2 is still column number 2 despite being moved.

## **ColSwap Event Applies To**

SSDBGrid

## **ColSwap Event See Also**

AfterPosChanged event

ColMove event

GrpMove event

GrpSwap event

## ColWidth Property

Applies To

### Description

Sets or returns the width of the column containing the currently selected button.

### Syntax

*object* . **ColWidth**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	A real number specifying the column width.

### Remarks

Valid range is 0 to 32767 with a default value of 0. Setting the value to 0 causes automatic computation based on the minimum column width.

This property affects all buttons in the column.

The unit of measurement is dictated by the form's **ScaleMode** property.

## **ColWidth Property Applies To**

DataOptSet

Close

## Collection Summary

Bookmarks

Buttons

Columns

Groups

SelBookmarks

StyleSets

## ColorMask Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the color of the **PictureButtons** bitmap, which will be interpreted as the background color.

### Syntax

*object* . **ColorMask**[= *color*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>color</i>	A long integer value or constant that determines the color of the PictureButtons bitmap.

### Remarks

The **ColorMask** property represents the color of the segmented button bitmap which will be interpreted as the background color. Each pixel of this color will be converted to the **BevelColorFace** color.

This property has no effect on the standard bitmap supplied with the control (when **PictureButtons** = None).

**ColorMaskEnabled** must be activated for this property to affect the control.

## **ColorMask Property Applies To**

SSDBData



## **ColorMask Property See Also**

[ColorMaskEnabled](#)

## ColorMaskEnabled Property

[See Also](#)

[Applies To](#)

### Description

Determines whether the **ColorMask** property will affect the active control.

### Syntax

*object* . **ColorMaskEnabled**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the <b>ColorMask</b> property is enabled, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	The <b>ColorMask</b> property will affect the control.
<b>False</b>	(Default) The <b>ColorMask</b> property will have no effect on the control.

### Remarks

This property has no effect on the standard bitmap supplied with the control (when **PictureButtons** = None).

## **ColorMaskEnabled Property Applies To**

SSDBData

**ColorMaskEnabled Property See Also**

[ColorMask](#)

## Cols Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the number of columns in the control.

### Syntax

*object* . **Cols**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the number of columns in the control.

### Remarks (SSDBCombo/SSDBDropDown/SSDBGrid)

At runtime, **Cols** is read-only.

At design time, this property determines the amount of columns to display in Unbound or AddItem modes. When working in bound mode, this property is automatically set, deriving the information from the database.

Remember that columns begin numbering at 0 (i.e., Columns(0) would actually be the first column).

### Remarks (SSDBOptSet)

For the SSDBOptSet control, the valid range is 1 to 10 with a default value of 1.

## **Cols Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

SSDBOptSet

## **Cols Property See Also**

[Col](#)

[Row](#)

[Rows](#)

**Column Header**



## Column Object

[See Also](#)

[Applies To](#)

### Description

The column object represents a column that is in a grid.

### Properties

---

<a href="#">Alignment</a>	<a href="#">HasBackColor</a>	<a href="#">Nullable</a>
<a href="#">AllowSizing</a>	<a href="#">HasForeColor</a>	<a href="#">NumberFormat</a>
<a href="#">BackColor</a>	<a href="#">HasHeadBackColor</a>	<a href="#">Position</a>
<a href="#">ButtonsAlways</a>	<a href="#">HasHeadForeColor</a>	<a href="#">Selected</a>
<a href="#">Caption</a>	<a href="#">HeadBackColor</a>	<a href="#">Style</a>
<a href="#">CaptionAlignment</a>	<a href="#">HeadForeColor</a>	<a href="#">StyleSet</a>
<a href="#">Case</a>	<a href="#">HeadStyleSet</a>	<a href="#">TagVariant</a>
<a href="#">ColChanged</a>	<a href="#">ItemData *</a>	<a href="#">Text</a>
<a href="#">DataField</a>	<a href="#">Left</a>	<a href="#">Top</a>
<a href="#">DataType</a>	<a href="#">Level</a>	<a href="#">Value</a>
<a href="#">DropDownhWnd</a>	<a href="#">List</a>	<a href="#">VertScrollBar</a>
<a href="#">FieldLen</a>	<a href="#">ListCount</a>	<a href="#">Visible</a>
<a href="#">ForeColor</a>	<a href="#">Locked</a>	<a href="#">Width</a>
<a href="#">Group</a>	<a href="#">Name</a>	

\* The **ItemData** property applies only to column objects that have their **Style** property set to '3 - Combo Box.'

### Methods

---

<a href="#">AddItem</a>	<a href="#">CellValue</a>	<a href="#">RemoveItem</a>
<a href="#">CellStyleSet</a>	<a href="#">IsCellValid</a>	
<a href="#">CellText</a>	<a href="#">RemoveAll</a>	

## **Column Object Applies To**

Columns collection

## **Column Object See Also**

Columns collection

## ColumnHeaders Property

See Also

Applies To

### Description

Determines whether column headers will be displayed.

### Syntax

*object* . **ColumnHeaders**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether column headers will be displayed, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) Column headers will be displayed.
<b>False</b>	Column headers will not be displayed.

## **ColumnHeaders Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

**ColumnHeaders Property See Also**

[GroupHeaders](#)

## Columns Collection

[See Also](#)

[Applies To](#)

[Example](#)

### Description

The columns collection represents a group of column objects that comprise a grid.

### Properties

---

[Count](#)

[Item](#)

### Methods

---

[Add](#)

[Remove](#)

[RemoveAll](#)

## **Columns Collection Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid



## **Columns Collection See Also**

Column object

## Columns Method

Applies To

### Description

Returns column object at specified index.

### Syntax

*object* . **Columns**( [*Index* **As Variant**])

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Index</i>	A variant expression that can be either the column index number or string specifying the column name (i.e., "Employee Name").

### Remarks

When no index is specified the column object is returned.

## **Columns Method Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

### **Combo Box Column Style**

This is different than the SSDBCombo control. Setting the Style property allows for a combo box in the grid. You can populate this box through the Grid Editor or through code.

## **ComboCloseUp Event**

See Also

Applies To

### **Description**

Occurs when a combo box is closed up.

### **Syntax**

**Sub** control\_ **ComboCloseUp** ()

## **ComboCloseUp Event Applies To**

SSDBGrid

## **ComboCloseUp Event See Also**

[ComboDropDown](#) event

## ComboDropDown Event

[See Also](#)

[Applies To](#)

### Description

Occurs when a combo box is dropped down.

### Syntax

```
Sub control_ComboDropDown ()
```



## **ComboDropDown Event Applies To**

SSDBGrid

## **ComboDropDown Event See Also**

[ComboCloseUp](#) event

## ComboDroppedDown Property

Applies To

### Description

Sets or returns the combo box's dropdown state.

### Syntax

*object* . **ComboDroppedDown**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the combo box is dropped down, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	Combo box is dropped down.
<b>False</b>	(Default) Combo box is not dropped down.

### Remarks

Setting this property to **False** in an event will cause the combo box not to drop down.

## **ComboDroppedDown Property Applies To**

SSDBGrid

## Data Widgets Constants

### Alignment Constants

Constant	Value	Description
ssAlignToRight	0	Right justify
ssAlignToLeft	1	Left justify

### AlignmentButtonPicture Constants

Constant	Value	Description
ssAlignPictureLeftofText	0	Left of text
ssAlignPictureRightofText	1	Right of text
ssAlignPictureFitToCaption	2	Fit to caption
ssAlignPictureTile	3	Tile

### AlignmentCaption Constants

Constant	Value	Description
ssAlignmentCaptionLeftTop	0	Left justify - Top
ssAlignmentCaptionLeftMiddle	1	Left justify - Middle
ssAlignmentCaptionLeftBottom	2	Left justify - Bottom
ssAlignmentCaptionRightTop	3	Right justify - Top
ssAlignmentCaptionRightMiddle	4	Right justify - Middle
ssAlignmentCaptionRightBottom	5	Right justify - Bottom
ssAlignmentCaptionCenterTop	6	Centered - Top
ssAlignmentCaptionCenterMiddle	7	Centered - Middle
ssAlignmentCaptionCenterBottom	8	Centered - Bottom

### AlignmentPictureCommand Constants

Constant	Value	Description
ssAlignmentPictureLeftTop	0	Left justify - Top
ssAlignmentPictureLeftMiddle	1	Left justify - Middle
ssAlignmentPictureLeftBottom	2	Left justify - Bottom
ssAlignmentPictureRightTop	3	Right justify - Top
ssAlignmentPictureRightMiddle	4	Right justify - Middle
ssAlignmentPictureRightBottom	5	Right justify - Bottom
ssAlignmentPictureCenterTop	6	Centered - Top
ssAlignmentPictureCenterMiddle	7	Centered - Middle
ssAlignmentPictureCenterBottom	8	Centered - Bottom
ssAlignmentPictureLeftOfText	9	Left of caption text
ssAlignmentPictureRightOfText	10	Right of caption text

ssAlignmentPictureAboveText	11	Above caption text
ssAlignmentPictureBelowText	12	Below caption text
ssAlignmentPictureStretch	13	Stretch to fill
ssAlignmentPictureTile	14	Tile to fill

### BevelColorScheme Constants

Constant	Value	Description
ssBevelColorSchemeGray	0	Uses shades of gray color scheme
ssBevelColorSchemeSystem	1	Uses system colors
ssBevelColorSchemeCustom	2	Uses custom color scheme

### BevelType Constants

Constant	Value	Description
ssBevelTypeNone	0	No beveling
ssBevelTypeInset	1	Bevel appears inset
ssBevelTypeRaised	2	Bevel appears raised

### BookmarkDisplayStyle Constants

Constant	Value	Description
ssBookmarkDisplayLRA	0	Least Recently Added
ssBookmarkDisplayMRA	1	Most recently added
ssBookmarkDisplaySorted	2	Sorted

### BookmarkShowStyle Constants

Constant	Value	Description
ssBookmarkShowNone	0	No bookmark buttons shown
ssBookmarkShowAdd	1	Show Add button only
ssBookmarkShowClear	2	Show ClearAll button only
ssBookmarkShowGoto	3	Show Goto button only
ssBookmarkShowAddClear	4	Show Add and ClearAll buttons
ssBookmarkShowAddGoto	5	Show Add and Goto buttons
ssBookmarkShowClearGoto	6	Show ClearAll and Goto buttons
ssBookmarkShowAll	7	Show all bookmark buttons

### BorderStyle Constants

Constant	Value	Description
ssBorderStyleNone	0	No border
ssBorderStyleFixedSingle	1	Fixed single-pixel border

## CaptionAlignment Constants

Constant	Value	Description
ssCaptionAlignmentLeft	0	Left justify
ssCaptionAlignmentRight	1	Right justify
ssCaptionAlignmentCenter	2	Center

## CaptionAlignmentEx Constants

Constant	Value	Description
ssCapAlignLeftTop	0	Left justify - Top
ssCapAlignLeftMiddle	1	Left justify - Middle
ssCapAlignLeftBottom	2	Left justify - Bottom
ssCapAlignRightTop	3	Right justify - Top
ssCapAlignRightMiddle	4	Right justify - Middle
ssCapAlignRightBottom	5	Right justify - Bottom
ssCapAlignCenterTop	6	Centered - Top
ssCapAlignCenterMiddle	7	Centered - Middle
ssCapAlignCenterBottom	8	Centered - Bottom

## CaptionAlignmentPicture Constants

Constant	Value	Description
ssCaptionAlignmentPictureLeftTop	0	Left justify - Top
ssCaptionAlignmentPictureLeftMiddle	1	Left justify - Middle
ssCaptionAlignmentPictureLeftBottom	2	Left justify - Bottom
ssCaptionAlignmentPictureRightTop	3	Right justify - Top
ssCaptionAlignmentPictureRightMiddle	4	Right justify - Middle
ssCaptionAlignmentPictureRightBottom	5	Right justify - Bottom
ssCaptionAlignmentPictureCenterTop	6	Centered - Top
ssCaptionAlignmentPictureCenterMiddle	7	Centered - Middle
ssCaptionAlignmentPictureCenterBottom	8	Centered - Bottom
ssCaptionAlignmentPictureLeftOfCaption	9	Left of caption text
ssCaptionAlignmentPictureRightOfCaption	10	Right of caption text
ssCaptionAlignmentPictureAboveCaption	11	Above caption text
ssCaptionAlignmentPictureBelowCaption	12	Below caption text
ssCaptionAlignmentPictureFitToCaption	13	Stretch to fill
ssCaptionAlignmentPictureTile	14	Tile to fill

## Case Constants

<b>Constant</b>	<b>Value</b>	<b>Description</b>
ssCaseUnchanged	0	Text case will not change
ssCaseLower	1	Text will be in lowercase
ssCaseUpper	2	Text will be in UPPERCASE

### **CellNavigation Constants**

<b>Constant</b>	<b>Value</b>	<b>Description</b>
ssCellNavigationCell	0	Arrow keys used to change cells
ssCellNavigationEdit	1	Arrow keys used to edit current cell

### **ColumnCaptionAlignment Constants**

<b>Constant</b>	<b>Value</b>	<b>Description</b>
ssColCapAlignLeftJustify	0	Left justify
ssColCapAlignRightJustify	1	Right justify
ssColCapAlignCenter	2	Center
ssColCapAlignUseColumnAlignment3		Use alignment specified for Column cells

### **Criteria Constants**

<b>Constant</b>	<b>Value</b>	<b>Description</b>
ssCriteriaSoundex	-1	Soundex
ssCriteriaLT	1	Less Than
ssCriteriaLE	2	Less than or equal
ssCriteriaEQ	3	Equal
ssCriteriaGE	4	Greater than or equal
ssCriteriaGT	5	Greater than
ssCriteriaPartial	6	Partial

### **Database Action Constants**

<b>Constant</b>	<b>Value</b>	<b>Description</b>
ssFirst	0	Go to first record
ssPreviousPage	1	Goto record <page> records before
ssPrevious	2	Goto previous record
ssNext	3	Goto next record
ssNextPage	4	Goto record <page> records after
ssLast	5	Goto last record
ssSaveBookmark	6	Save current bookmark
ssGotoBookmark	7	Goto specified bookmark
ssRefresh	8	Refresh datasource



## DataMode Constants

Constant	Value	Description
ssDataModeBound	0	Control is in data bound mode
ssDataModeUnbound	1	Control is in unbound mode
ssDataModeAddItem	2	Control is in AddItem mode

## Direction Constants

Constant	Value	Description
ssDirectionUp	1	Search towards the first record
ssDirectionDown	2	Search towards the last record

## DividerStyle Constants

Constant	Value	Description
ssDividerStyleBlackline	0	Black line
ssDividerStyleDarkGrayline	1	Dark gray line
ssDividerStyleRaised	2	Raised
ssDividerStyleInset	3	Inset
ssDividerStyleForeColor	4	Use control's ForeColor

## DividerType Constants

Constant	Value	Description
ssDividerTypeNone	0	No divider
ssDividerTypeVertical	1	Vertical dividers
ssDividerTypeHorizontal	2	Horizontal dividers
ssDividerTypeBoth	3	Horizontal and vertical dividers

## FindShowStyle Constants

Constant	Value	Description
ssFindShowNone	0	No search buttons shown
ssFindShowFind	1	Show Find button
ssFindShowFindNext	2	Show Find and Find Next button.
ssFindShowAll	3	Show Find, Find Next and Find Previous buttons.

## Font3D Constants

Constant	Value	Description
ssFont3DNone	0	No 3-D font effect

ssFont3DRaisedLight	1	Raised letters w/light shading
ssFont3DRaisedHeavy	2	Raised letters w/heavy shading
ssFont3DInsetLight	3	Inset letters w/light shading
ssFont3DInsetHeavy	4	Inset letters w/heavy shading

## MousePointer Constants

Constant	Value	Description
ssMousePointerDefault	0	Default cursor
ssMousePointerArrow	1	Arrow cursor
ssMousePointerCross	2	Cross cursor
ssMousePointerIBeam	3	I-Beam cursor
ssMousePointerIcon	4	Icon cursor
ssMousePointerSize	5	Size cursor
ssMousePointerSizeNESW	6	Size NE-SW cursor
ssMousePointerSizeNS	7	Size N-S cursor
ssMousePointerSizeNWSE	8	Size NW-SE cursor
ssMousePointerSizeWE	9	Size W-E cursor
ssMousePointerUpArrow	10	Up Arrow (Alternate Select)
ssMousePointerHourglass	11	Hourglass
ssMousePointerNoDrop	12	No drop
ssMousePointerArrowAndHourGlass	13	Arrow & Hourglass
ssMousePointerArrowAndQuestion	14	Arrow & Question
ssMousePointerSizeAll	15	Size All
ssMousePointerCustom	99	Custom

## Nullable Constants

Constant	Value	Description
ssColumnNullableAutomatic	0	Control will determine how to store null values
ssColumnNullableNull	1	Null values will be stored as nulls
ssColumnNullableEmptyString	2	Null values will be stored as empty strings.

## OrientationStyle Constants

Constant	Value	Description
ssOrientationHorizontal	0	Control will be horizontal
ssOrientationVertical	1	Control will be vertical

## ReadType Constants

Constant	Value	Description
----------	-------	-------------

ssReadTypeAllData	0	Read all data from record source
ssReadTypeBookmarkOnly	1	Read bookmarks only from recordsource
ssReadTypeBookmarkAndBoundColumn	2	Read bookmarks and contents of bound column from recordsource
ssReadTypeBookmarkAndDisplayColumn	3	Read bookmarks and contents of displayed column from recordsource

## Relocate Constants

Constant	Value	Description
ssRelocateNotAllowed	0	Columns cannot be moved
ssRelocateWithinGroup	1	Columns may be re-arranged within a group
ssRelocateAnywhere	2	Columns may be moved anywhere in the control

## RowNavigation Constants

Constant	Value	Description
ssRowNavigationFull	0	Full movement using arrow keys
ssRowNavigationLRLock	1	Left/Right movement only within row
ssRowNavigationUDLock	2	Up/Down movement only within row
ssRowNavigationAllLock	3	All movement locked by row

## RowSelectionStyle Constants

Constant	Value	Description
ssRowSelectionStyleListBox	0	Listbox style
ssRowSelectionStyleInvert	1	Invert colors
ssRowSelectionStyle3D	2	3-D effect

## Scale Constants

Constant	Value	Description
ssScaleTwips	0	Twips
ssScalePixels	1	Pixels
ssScaleHiMetric	2	HiMetric
ssScaleContainer	3	Container

## ScrollBarsStyle Constants

Constant	Value	Description
ssScrollBarsNone	0	No scroll bars
ssScrollBarsHorizontal	1	Horizontal scroll bar only
ssScrollBarsVertical	2	Vertical scroll bar only

ssScrollBarsBoth	3	Horizontal and vertical scroll bars
ssScrollBarsAutomatic	4	Scroll bars appear as needed

## SelectionType Constants

Constant	Value	Description
ssSelectionTypeNone	0	No selection allowed
ssSelectionTypeSingleSelect	1	Select single item only
ssSelectionTypeMultiSelect	2	Select multiple items individually
ssSelectionTypeMultiSelectRange	3	Select multiple items in a range

## SelType Constants

Constant	Value	Description
ssSelTypeGroup	0	Select by group
ssSelTypeColumn	1	Select by column
ssSelTypeRow	2	Select by row

## Style Constants

Constant	Value	Description
ssStyleEdit	0	Edit type cell
ssStyleEditButton	1	Edit type cell with button (ellipsis)
ssStyleCheckBox	2	Check box type cell
ssStyleComboBox	3	Combo box type cell
ssStyleButton	4	Button type cell

## TabNavigation Constants

Constant	Value	Description
ssMoveToNextCell	0	Tab key moves focus to next cell
ssMoveToNextControl	1	Tab key moves focus to next control

## WhatChanged Constants

Constant	Value	Description
ssWhatChangedColMoved	0	Column was moved
ssWhatChangedGrpMoved	1	Group was moved
ssWhatChangedColSwapped	2	Column was swapped
ssWhatChangedGrpSwapped	3	Group was swapped

## WhereIs Constants

ssWhereIsNothing	0	Cursor points at nothing
ssWhereIsGridHeading	1	Cursor points at grid heading
ssWhereIsGroupHeading	2	Cursor points at group header
ssWhereIsColumnHeading	3	Cursor points at column header
ssWhereIsGridSelector	4	Cursor points at grid selector (left of group & column headers)
ssWhereIsRecordSelectors	5	Cursor points at a record selector
ssWhereIsBackground	6	Cursor points at background area
ssWhereIsData	7	Cursor points at grid cells

### WhereIsDataOptSet Constants

Constant	Value	Description
ssWhereIsNothing	0	Cursor points at nothing
ssWhereIsButton	1	Cursor points at a button
ssWhereIsCaption	2	Cursor points at a caption

### WhereIsEnhancedDataControl Constants

ssWhereIsNothing	0	Cursor points at nothing
ssWhereIsCaption	1	Cursor points at caption
ssWhereIsBevel	2	Cursor points at bevel
ssWhereIsButtonFirst	3	Cursor points at First Record button
ssWhereIsButtonLast	4	Cursor points at Last Record button
ssWhereIsButtonPrevPage	5	Cursor points at Previous Page button
ssWhereIsButtonNextPage	6	Cursor points at Next Page button
ssWhereIsButtonPrev	7	Cursor points at Previous Record button
ssWhereIsButtonNext	8	Cursor points at Next Record button
ssWhereIsButtonAdd	9	Cursor points at Add Record button
ssWhereIsButtonCancel	10	Cursor points at Cancel Edits button
ssWhereIsButtonUpdate	11	Cursor points at Update Edits button
ssWhereIsButtonDelete	12	Cursor points at Delete Record button
ssWhereIsButtonFindNext	13	Cursor points at Find Next button
ssWhereIsButtonFindPrev	14	Cursor points at Find Previous button
ssWhereIsButtonFind	15	Cursor points at Find button
ssWhereIsButtonBookmarkAdd	16	Cursor points at Add Bookmark button
ssWhereIsButtonBookmarkClear	17	Cursor points at Clear All Bookmarks button
ssWhereIsButtonBookmarkGoto	18	Cursor points at Goto Bookmark button





Copyright Notice

New! **2.0**  
**Data**



### **What's New?**

A look at what's new in Data Widgets 2.0. Includes information on Version 1.0 to 2.0 conversion issues.

### **Guided Tours**

Designed to get you up and running quickly by walking you through Data Widgets.



### **Frequently Asked Questions**

Concise answers to the questions most often asked about how to use Data Widgets.



### **How-To Help Contents**

Task-based, step by step guidance on how to perform common tasks using Data Widgets.



### **Control Descriptions**

Explains what Data Widgets is, and describes each of the Data Widgets custom controls, giving detailed information on each.

Properties  
Events  
Methods

### **Control Reference**

An alphabetical listing of all programming language topics

#### **Properties**

#### **Events**

#### **Methods**

#### **Objects**

#### **Collections**

#### **Errors**

#### **Constants**

#### **Extra Samples**



### **Technical Information / Distribution Notes**

Specifications such as system requirements, included files, files needed for distribution, and error messages., plus performance tuning information



### **Technical Support**

Getting technical and product support for Sheridan products.

### **Control Caption Area**



### **Control Descriptions**



### **What is Data Widgets?**

Background on the product and its features as well as information on how to use Data Widgets with your applications.



### **Data Grid**

A fully editable bound grid that allows you to edit an entire record set, regardless of size, on screen without writing any code. Also supports Unbound and AddItem modes.



### **Data Combo**

A bound combo box you can include in your application.



### **Data DropDown**

A control used for attaching a Data Grid column to a dropdown list of values from another source of data.



### **DataOptionSet**

Option buttons that can be used to represent field values by binding the control to a data source.



### **Enhanced Data Control**

A front end to the Visual Basic data control adding new functionality such as bookmark navigation and page movement.



### **Data Command Button**

Command buttons that perform database functions.

Copyright © 1993-1997 Sheridan Software Systems, Inc. All rights reserved.

Information in this document is subject to change without notice and does not represent a commitment on the part of Sheridan Software Systems. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy the software on any medium except as specifically allowed in the license or nondisclosure agreement. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the express written permission of Sheridan Software Systems, Inc.

Data Widgets, DataGrid, and the Sheridan logo are trademarks of Sheridan Software Systems, Inc.

Microsoft, Visual Basic and Windows are registered trademarks of Microsoft Corporation.

All other trademarks and registered trademarks are the property of their respective owners.



## Count Property

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Returns the total number of objects in the specified collection.

### Syntax

*object* . **Count**

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.

### Remarks

The **Count** property is used only to return the number of objects in a collection. To set the number of objects, use the **Add** and **Remove** methods.

## **Count Property Applies To**

Bookmarks collection

Buttons collection

Columns collection

Groups collection

SelBookmarks collection

StyleSets collection

## **Count Property See Also**

Bookmarks collection

Buttons collection

Columns collection

Groups collection

StyleSets collection

Add method

Remove method

RemoveAll method

## Creating Buttons at Design Time

To create DataOptionSet buttons at runtime:

1. Select the total number of buttons by setting the **NumberOfButtons** property. The control will immediately redraw to reflect the new setting.
2. Set the **IndexSelected** property to the button number you want to modify. All changes made to button-specific properties will affect the button selected with this property.
3. Set the **OptionValue** property for this button. This is the value that will be compared against the database value.
4. Set any additional properties you wish to alter.
5. Repeat Steps 2-4 as needed.

## Creating Buttons at Runtime

Creating buttons at runtime is a much simpler task thanks to the Button Object and its related Buttons Collection. The Button Object makes it possible to directly access a button without the need for selecting the IndexSelected property first. Additionally, you do not need to set the NumberOfButtons property since adding or deleting a button object within the collection automatically updates this value.

For example, to modify the fifth button's caption, you only need to write the code:  
`SSDBOptSet1.Buttons(4).Caption = "Fifth Button".`

## Creating a Total Query

This example makes use of the **Count** and **Sum** statements of SQL. This example can be used with any data source that accepts SQL query statements as a valid recordsource.

This example uses a database that contains information about products and orders. The totals provided by the query include the total quantity of orders placed and the total dollar amount of all orders. The query also returns the total number of records.

To set up the application, you need two data sources (such as the Visual Basic Data Control.) The first data source should connect directly to the *Order Details* table in the database. Order Details contains data about individual orders, including the unique ID that identifies the order (Order ID), the ID code of the product ordered (Product ID), the quantity of products ordered (Quantity), and the unit price of the product (Unit Price).

In Visual Basic, you can simply choose the table name from the list of available RecordSources in the database. Or you could specify an SQL statement for the recordsource (The query must appear as one line of text; line continuation [underscore] characters are used here for readability but should be omitted from the actual query statement.):

```
SELECT [Order ID], [Product ID], [Quantity],_  
[Unit Price] FROM [Order Details];
```

This is the recordsource that will be used to populate the DataGrid.

### The Total Query

To create the second recordsource, specify the following SQL query (Again, underscore characters should be omitted from the actual query statement.):

```
SELECT Count([Order ID]) as [Total Orders],_  
Sum([Unit Price]) As [Total Price],_  
Sum([Quantity]) as [Total Quantity] FROM [Order Details];
```

The controls displaying the totals would then be bound to the fields in the second recordsource that are created by the total query. The control displaying the total number of records will be bound to the "Total Orders" field. The control displaying the total price will be bound to the "Total Price" field, and the control displaying the total quantity will be bound to the "Total Quantity" field.

A separate control can be used to display the product of the "Total Quantity" and the "Total Price" field to display the total dollar amount of all orders.

Data Widgets 2.0 Development Team	
Task	Team Member
Product Manager	NedR
Development	BILLB
Development	MaryS
Development	NickC
Development	RobS
Development	TonyA
Docs/Help	DaveP & JimD
QA	GaryD
QA	JasonM
QA	SaraK
QA	VinnyP
Tech Support	DanW

## Customizing the Bound Data Combo

When you select a data combo, all fields in the associated database are displayed by default. Sometimes, this is not a convenient way of displaying your data. Using the **Visible** property on individual columns allows you to display only the field you want listed.

For example, the following code displays only the first and third columns of a database with four fields:

```
Sub SSDBCombo1_InitColumnProps()  
SSDBCombo1.Columns(1).Visible = False  
SSDBCombo1.Columns(3).Visible = False  
End Sub
```

For simplicity, you could use the [Grid Editor](#) to make the changes.



## Customizing the Bound Data DropDown

When you select a Data DropDown, all fields in the associated database are displayed by default. Sometimes, this is not a convenient way of displaying your data. Using the **Visible** property on individual columns allows you to display only the field you want listed. Refer to the Properties listing for a complete listing of properties used with the Data DropDown control.

For example, the following code displays only the first and third columns of a database with four fields:

```
Sub SSDBDropDown1_InitColumnProps()  
    SSDBDropDown1.Columns(1).Visible = False  
    SSDBDropDown1.Columns(3).Visible = False  
End Sub
```

For simplicity, you could use the [Grid Editor](#) to make the changes.

[◀ Back](#)

## **Data Combo Collections**

[Columns](#)

[Groups](#)

[SelBookmarks](#)

[StyleSets](#)



## **Data Combo Control**

[Properties](#)

[Events](#)

[FileNames](#)

[Methods](#)

[ObjectType](#)

[Collections](#)

[Objects](#)

### Anatomy of a Data Combo

The Data Combo custom control is a combo box that can be used to display/edit a field value from one record set while providing a dropdown list of field values from another set. The Data Combo functions in bound, unbound and AddItem modes.

For bound mode operation, you simply need to set four properties; two for the edit portion and two for the list portion. When you dropdown the list, it will automatically be filled with the rows and columns of the record sets you chose.

### Keyboard Interface

[Adding a Bound Data Combo](#)

[Customizing the Bound Data Combo](#)

[Adding an Unbound Data Combo](#)

[Achieving a 3D Look with the Data Combo](#)

[Binding to a Data Control Across Forms](#)

[◀ Back](#)

## **Data Combo Events**

[Change](#)

[Click](#)

[CloseUp](#)

[DbClick](#)

[DragDrop](#)

[DragOver](#)

[DropDown](#)

[GotFocus](#)

[InitColumnProps](#)

[KeyDown](#)

[KeyPress](#)

[KeyUp](#)

[LostFocus](#)

[MouseDown](#)

[MouseMove](#)

[MouseUp](#)

[PositionList](#)

[RowLoaded](#)

[Scroll](#)

[ScrollAfter](#)

[TextError](#)

[UnboundPositionData](#)

[UnboundReadData](#)

[ValidateList](#)

## **Data Combo Features**

The Data Combo is a bound combo box you can include in your application.

The following is a list of the features found in this control:

- Variable edit area height similar to that used in Microsoft Access

- Multiline edit area

- User is not limited to the width of the edit area for entering/displaying data

- Same formatting capabilities as the SSDBGrid control

- Full design time capabilities

[◀ Back](#)

## **Data Combo Methods**

[AddItem](#)

[AddItemBookmark](#)

[AddItemRowIndex](#)

[ColContaining](#)

[Columns](#)

[DoClick](#)

[Drag](#)

[GetBookmark](#)

[Groups](#)

[GrpContaining](#)

[IsItemInList](#)

[IsTextValid](#)

[MoveFirst](#)

[MoveLast](#)

[MoveNext](#)

[MovePrevious](#)

[MoveRecords](#)

[ReBind](#)

[Refresh](#)

[RemoveAll](#)

[RemoveItem](#)

[Reset](#)

[RowBookmark](#)

[RowContaining](#)

[RowTop](#)

[Scroll](#)

[SelBookmarks](#)

[StyleSets](#)

[◀ Back](#)

## **Data Combo Objects**

[Column](#)

[Font](#)

[Group](#)

[HeadFont](#)

[ssRowBuffer](#)

[StyleSet](#)

[◀ Back](#)

## **Data Combo Properties**

[\(About\)](#)

[\(Custom\)](#)

[Align](#)

[AllowInput](#)

[AllowNull](#)

[AutoRestore](#)

[BackColor](#)

[BackColorEven](#)

[BackColorOdd](#)

[BevelColorFace](#)

[BevelColorFrame](#)

[BevelColorHighlight](#)

[BevelColorScheme](#)

[BevelColorShadow](#)

[BevelType](#)

[BevelWidth](#)

[Bookmark](#)

[BorderStyle](#)

[CheckBox3D](#)

[Cols](#)

[ColumnHeaders](#)

[DataChanged](#)

[DataField](#)

[DataFieldList](#)

[DataFieldToDisplay](#)

[DataMode](#)

[DataSource](#)

[DataSourceList](#)

[DefColWidth](#)

[DividerStyle](#)

[DividerType](#)

[DragIcon](#)

[DragMode](#)

[DroppedDown](#)

[Enabled](#)

[FieldDelimiter](#)

[FieldSeperator](#)

[FirstRow](#)

[Font](#)

[Font3D](#)



ForeColor  
ForeColorEven  
ForeColorOdd  
GroupHeaders  
GroupHeadlines  
HeadFont  
HeadFont3D  
Headlines  
HeadStyleSet  
Height  
HelpContextID  
hWnd  
Index  
Left  
LeftCol  
LeftGrp  
LevelCount  
ListAutoPosition  
ListAutoValidate  
ListWidth  
ListWidthAutoSize  
MaxDropDownItems  
MinDropDownItems  
MouseIcon  
MousePointer  
MultiLine  
Name  
Negotiate  
PictureDropDown  
Redraw  
Row  
RowHeight  
Rows  
RowSelectionMode  
Scrollbars  
SelLength  
SelStart  
SelText  
StyleSet  
TabIndex  
TabStop  
Tag

TagVariant

Text

TextFormat

Top

UseExactRowCount

Visible

VisibleCols

VisibleGrps

VisibleRows

WhatsThisHelpID

Width

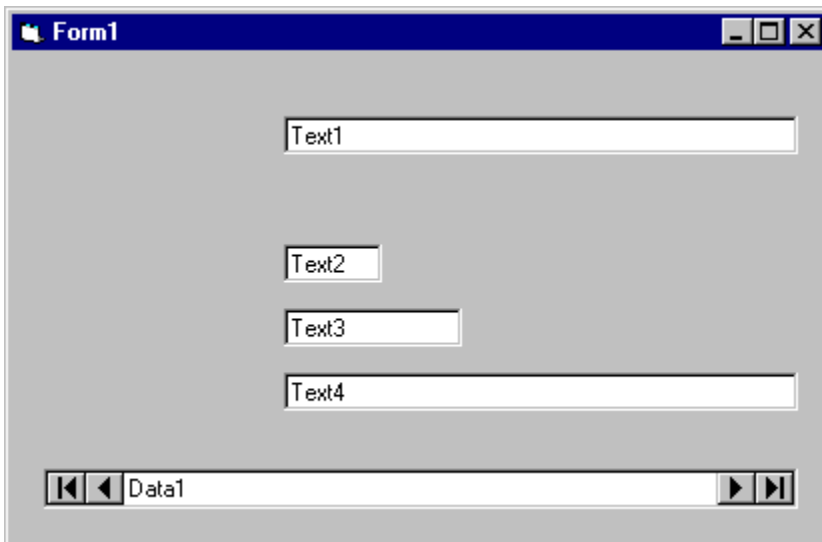
## Data Combo: Exercise 1

This section guides you through the creation of a sample program using the Data Combo control. For a complete description of this control, refer to the [Data Combo Control](#)

For the exercises in this section, it is assumed that you have already launched your development application (Visual Basic), and that the control has been added to your toolbox. For more information on how to do this, refer to [Using Data Widgets](#).

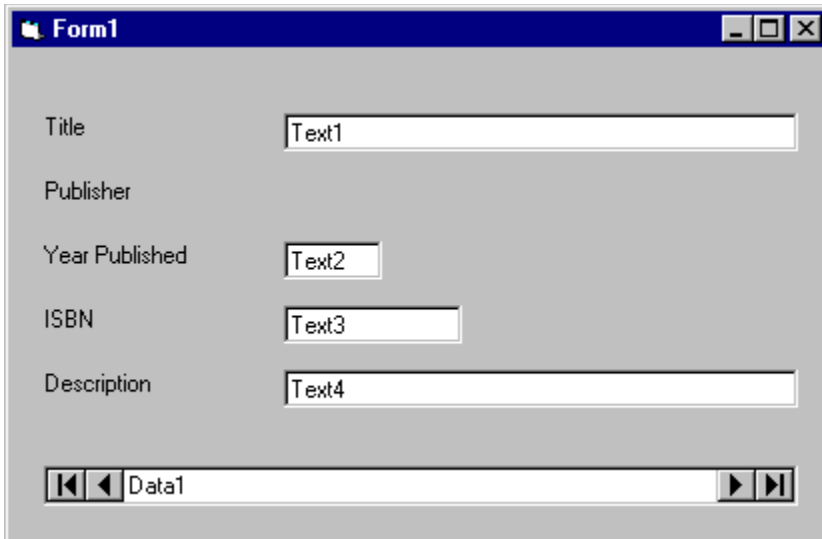
In this exercise, you will create a data entry program that makes use of the Data Combo control.

1. Place a Visual Basic data control on the form.
2. Set the **DatabaseName** property to BIBLIO.MDB. Be sure to qualify the filename with the path of where the file is located, if needed.
3. Set the **RecordSource** property to 'Titles'.
4. Place four text boxes on the form. Your form should look like this:



The screenshot shows a window titled 'Form1' with a grey background. It contains four text boxes arranged vertically: 'Text1' (wide), 'Text2' (narrow), 'Text3' (medium), and 'Text4' (wide). At the bottom, there is a Data Combo control labeled 'Data1' with navigation arrows on either side.

5. Create five text labels on the form. Your form should look like this:



The screenshot shows a window titled 'Form1' with a grey background. It contains five text labels on the left side: 'Title', 'Publisher', 'Year Published', 'ISBN', and 'Description'. To the right of these labels are four text boxes: 'Text1' (wide), 'Text2' (narrow), 'Text3' (medium), and 'Text4' (wide). At the bottom, there is a Data Combo control labeled 'Data1' with navigation arrows on either side.


6. For each text box, set the **DataSource** field to *Data1* and the **DataField** as follows:

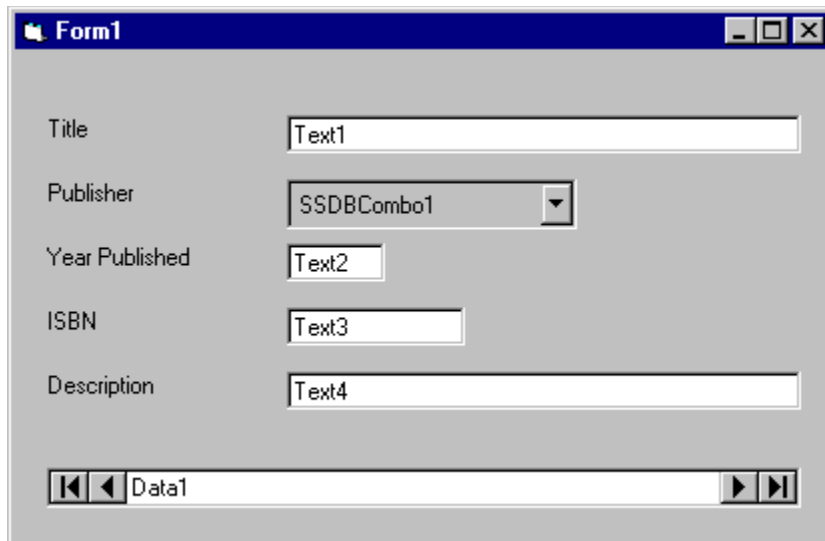
**Text1.DataField** = 'Title'

**Text2.DataField** = 'Year Published'

**Text3.DataField** = 'ISBN'

**Text4.DataField** = 'Description'

7. Add a SSDBCombo  control to the form. Your form should now look like this:



The screenshot shows a window titled "Form1" with a standard Windows-style title bar. Inside the window, there are five text boxes arranged vertically. The first is labeled "Title" and contains the text "Text1". The second is labeled "Publisher" and is an SSDBCombo control showing "SSDBCombo1" with a dropdown arrow. The third is labeled "Year Published" and contains "Text2". The fourth is labeled "ISBN" and contains "Text3". The fifth is labeled "Description" and contains "Text4". At the bottom of the form is a Data DropDown control with a list box containing "Data1" and navigation buttons (back, forward, first, last).

8. Place a second Visual Basic data control on the form.

9. Set the data control's **DatabaseName** property to BIBLIO.MDB.

10. Set the data control's **RecordSource** property to 'Publishers'.

11. Set the data control's **Visible** property to 'False'.

12. Set the data combo's **DataSource** property to 'Data1'. This binds the edit portion of the data combo to the 'Titles' table.

13. Set the data combo's **DataField** property to 'PubID'. This determines the field used in the edit portion of the data combo.

14. Set the data combo's **DataSourceList** property to 'Data2'. This binds the list portion of the data combo to the 'Publishers' table.

15. Set the data combo's **DataFieldList** property to 'PubID'. This determines the field used in the list portion of the data combo.

Try running your application at this point, using the data control to navigate through the records. Click on the Data DropDown and you'll see how the record in the edit field is automatically selected in the list portion. Try changing the field value by selecting another record from the list.

This is a quick example of how the Data Combo can be used in your applications. Save this project, as we'll be using it in the next exercise.

## Data Combo: Exercise 2

This exercise demonstrates how you can customize the Data Combo to suit your specific needs. This exercise makes use of objects and collections. If you are not familiar with object and collections, you should refer to the Introduction to OCX Controls section before proceeding. The project used in the last exercise should be running at this point.

As you saw in the last exercise, when you click on the data combo, the entire table in the list portion displays. Sometimes this is fine, but there are times when you want to limit what is displayed and how it is displayed. The Data Combo allows you to make use of the objects that the Data Grid uses.

Let's make it so that the Data Combo only displays the fields "PubID", "Name", and "Company Name". In the **SSDBCombo1\_DropDown** event, add the following code:

```
SSDBCombo1.Columns(3).Visible = False
SSDBCombo1.Columns(4).Visible = False
SSDBCombo1.Columns(5).Visible = False
SSDBCombo1.Columns(6).Visible = False
SSDBCombo1.Columns(7).Visible = False
SSDBCombo1.Columns(8).Visible = False
SSDBCombo1.Columns(9).Visible = False
```

Run your application, and drop down the Data Combo. You will now see that only the PubID, Name, and Company Name fields appear.

Altering properties is a simple task when dealing with objects. If we wanted to make the first column a different color, all we need to add in the **SSDBCombo1\_DropDown** event is the following:

```
SSDBCombo1.Columns(0).BackColor=RGB(200,200,200)
```

Try experimenting with setting different properties on the control, as well as on the columns themselves. Remember that you can also use the Grid Editor to customize the Data DropDown properties.



## Data Command Button Control

FileNames

ObjectType

[Properties](#)

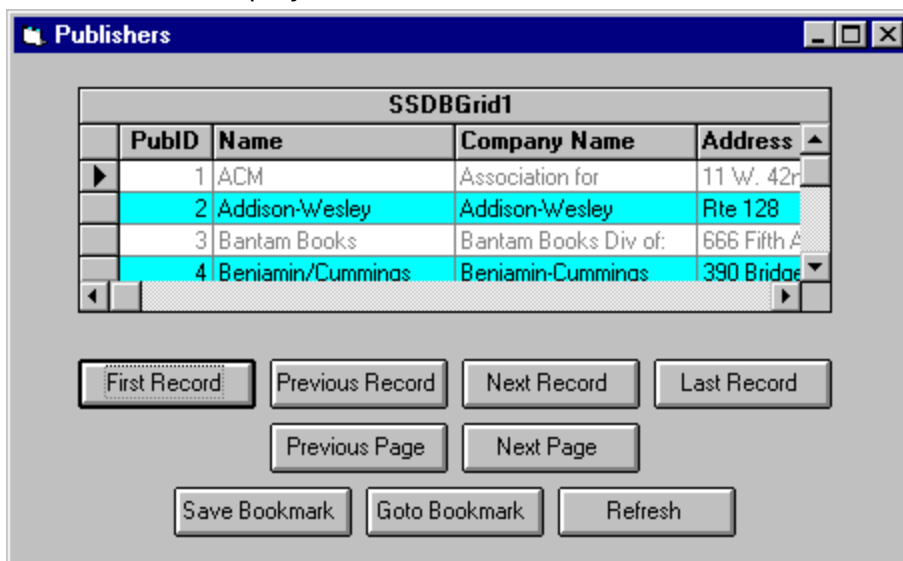
[Events](#)

[Methods](#)

The Data Command custom control is a 3D command button that can be bound to a data control. This button can be configured to automatically perform database actions as designated by you when clicked. The control also contains a speed button feature, allowing the user to hold the button in the down state to repeat a function.

A full set of appearance properties, including the capability of placing pictures on the buttons, are available with the Data Command control. The command button can be set to perform one of the following database actions:

- Goto first record
- Goto previous record
- Goto next record
- Goto last record
- Goto previous page (pages are defined by you)
- Goto next page (pages are defined by you)
- Create bookmark
- Goto bookmark
- Refresh display



[Adding a Data Command Button](#)

[Speed Buttons](#)

[Binding to a Data Control Across Forms](#)

[◀ Back](#)

## **Data Command Button Events**

[AfterClick](#)

[Click](#)

[DragDrop](#)

[DragOver](#)

[GotFocus](#)

[KeyDown](#)

[KeyPress](#)

[KeyUp](#)

[LostFocus](#)

[MouseDown](#)

[MouseMove](#)

[MouseUp](#)

## **Data Command Button Features**

The Data Command Button allows you to create command buttons that perform database functions.

The following is a list of the features for this control:

- Add, Delete, Refresh, Bookmark, and Auto-Positioning functions

- Click and After Click events

- Auto-Repeat functionality

- 3D font capability

- Multiline captions

- Custom color options

- Auto-sizing capability



[◀ Back](#)

## **Data Command Button Methods**

[DoClick](#)

[Drag](#)

[Refresh](#)

[◀ Back](#)

## **Data Command Button Properties**

[\(About\)](#)

[\(Custom\)](#)

[AutoSize](#)

[BevelColorFace](#)

[BevelColorFrame](#)

[BevelColorHighlight](#)

[BevelColorScheme](#)

[BevelColorShadow](#)

[BevelWidth](#)

[Cancel](#)

[Caption](#)

[CaptionAlignment](#)

[DatabaseAction](#)

[DataSource](#)

[Default](#)

[DelayInitial](#)

[DelaySubsequent](#)

[DragIcon](#)

[DragMode](#)

[Enabled](#)

[Font](#)

[Font3D](#)

[ForeColor](#)

[Height](#)

[HelpContextID](#)

[hWnd](#)

[Index](#)

[Left](#)

[MouseIcon](#)

[MousePointer](#)

[Name](#)

[PageValue](#)

[Picture](#)

[PictureAlignment](#)

[PictureMetaHeight](#)

[PictureMetaWidth](#)

[RoundedCorners](#)

[SavedBookmark](#)

[TabIndex](#)

[TabStop](#)

Tag

TagVariant

Top

Visible

WhatsThisHelpID

Width

WordWrap

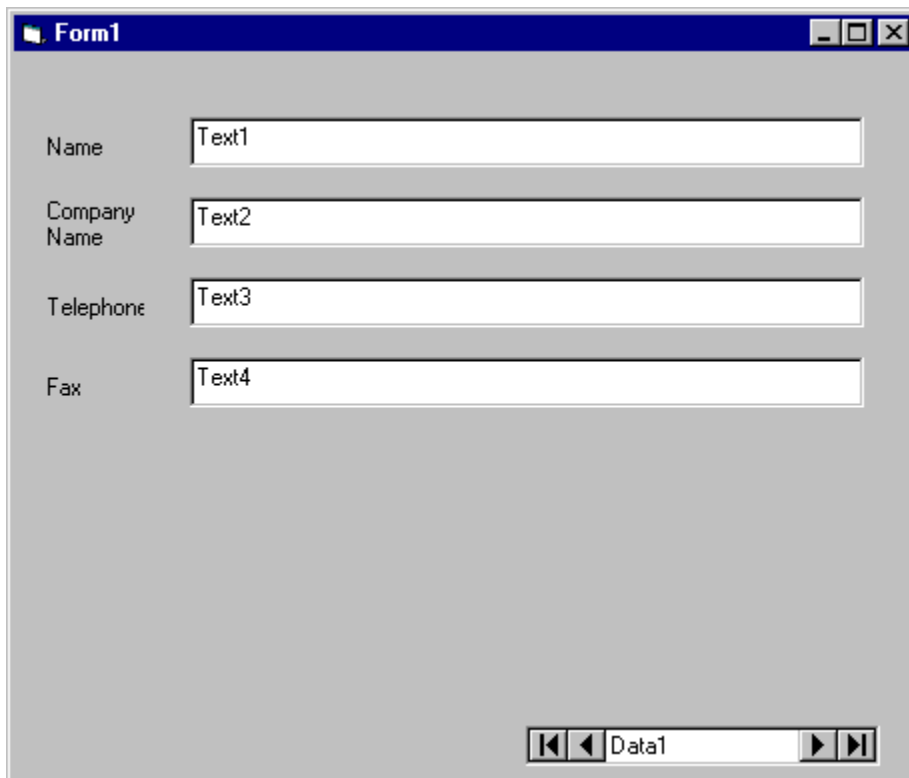
## Data Command: Exercise 1

This guides you through the creation of a sample program using the Data Command control. For a complete description of this control, refer to the [Data Command](#) section.

For this exercise, it is assumed that you have already launched your development application (Visual Basic), and that the control has been added to your toolbox. For more information on how to do this, refer to [Using Data Widgets](#).

You will create an application that makes use of the Data Command control. The database used will be the BIBLIO.MDB that ships with Visual Basic and is located in the Visual Basic home directory. If you are using an environment other than Visual Basic, and do not have the BIBLIO.MDB database, consult [BIBLIO File Structure](#) for details on the file layout.

1. Place a standard data control on the form.
2. Set the **DatabaseName** property to BIBLIO.MDB and set the **RecordSource** property to 'Publishers'. Set the **Visible** property to 'False'.  
This example will demonstrate how the Data Command buttons can replace navigation functions of the data control. We can set the data control to invisible since we will use the buttons for navigation.
3. Add four standard labels and four standard text boxes as shown below. Set the **DataSource** property to 'Data1' for all text boxes.
4. Set the **DataField** for the first text box to 'Name', set the second one to 'Company Name', the third to 'Telephone', and the fourth to 'Fax'.

The image shows a screenshot of a Visual Basic form window titled "Form1". The form has a light gray background and a blue title bar. It contains four text boxes arranged vertically, each with a label to its left: "Name" (Text1), "Company Name" (Text2), "Telephone" (Text3), and "Fax" (Text4). At the bottom center of the form, there is a Data Command control. This control consists of a small rectangular area with a white background and a gray border. Inside this area, there are four navigation buttons: a double-left arrow, a single-left arrow, a single-right arrow, and a double-right arrow. To the right of these buttons, the text "Data1" is displayed.

5. Add six Data Command buttons to the form setting the **DataSource** property for each one to 'Data1' and changing the **Caption** property for each one so that they look like this:



6. For each button, set the **DatabaseAction** property so it corresponds to the caption.
7. The Save Bookmark button requires one line of code to make it functional. To add it, double click on the Save Bookmark button and add the following:

```
Private Sub SSDBCommand3_AfterClick()  
    SSDBCommand6.SavedBookmark = SSDBCommand3.SavedBookmark  
End Sub
```

8. Run the application.

Screenshot of a Windows application window titled "Form1". The window contains four text boxes for "Name", "Company Name", "Telephone", and "Fax". The "Name" box contains "ACM", "Company Name" contains "Association for Computing Machinery", and "Telephone" contains "212-869-7440". Below the text boxes are six buttons: "First Record", "Last Record", "Previous Record", "Next Record", "Save Bookmark", and "Goto Bookmark".

And there you have it! This is just a sampling of what the Data Command buttons are capable of. Try using this application to get a feel for how the buttons work. To see how the bookmarks work, click the Save Bookmark button, and then move to another record. Clicking on the Goto Bookmark button will take you back to the record you saved.

[◀ Back](#)

## **Data DropDown Collections**

[Columns](#)

[Groups](#)

[SelBookmarks](#)

[StyleSets](#)



## Data DropDown Control

[Properties](#)

[Events](#)

[Methods](#)

[FileNames](#)

[ObjectType](#)

[Collections](#)

[Objects](#)

The Data DropDown custom control is a grid that can be linked to the cells in the [Data Grid](#) (SSDBGrid) for use as a value selection list. The Data DropDown, when used in conjunction with a cell in a Data Grid, functions very similar to the Data Combo, with the exception that it does not contain an edit portion. The field that would normally be in the edit portion of a Data Combo is in the cell of the Data Grid. You can display as few or as many fields in the dropdown list as you want.

One of the advantages of the Data DropDown is that it allows you to cross-reference data to a value. Let's say you have a field (**EmployeeID**) that stores the identification number of your employees. Instead of a person needing to memorize each employee's number, you can create a Data DropDown in the **EmployeeID** field that lists the employee's full name next to their identification number in a scrollable list for easy selection. You could do the same for customers, parts, or just about any other information that you want to access from a list.

Order Lookup Table			
EmployeeID	Date	Customer Name	Customer Address
10	10/10/95	General Savings	109 Plymouth
15	10/10/95	Amplified Elec.	10-20 82 Street
EmployeeID	EmployeeName	ank	210 East 32nd
15	Mark Ronstein		982 Woodside
4	John Ewens		222 East 60th
9	Richard Adams		10291 East 10
1	Bob Sanderson		

### [Keyboard Interface](#)

### [Adding a Bound Data DropDown](#)

### [Customizing the Bound Data DropDown](#)

### [Adding an Unbound Data DropDown](#)

### [Binding to a Data Control Across Forms](#)

[◀ Back](#)

## **Data DropDown Events**

[Click](#)

[CloseUp](#)

[DragDrop](#)

[DragOver](#)

[DropDown](#)

[InitColumnProps](#)

[PositionList](#)

[RowLoaded](#)

[Scroll](#)

[ScrollAfter](#)

[TextError](#)

[UnboundPositionData](#)

[UnboundReadData](#)

[ValidateList](#)



## **Data DropDown Features**

The Data DropDown control is used for attaching a Data Grid column to a dropdown list of values from another source of data.

The following is a list of the features for this control:

- Used in conjunction with the Data Grid

- Supports multiple data modes including bound, unbound, and AddItem

- User is not limited to the width of the edit area for entering/displaying data

- Same formatting capabilities as the SSDBGrid control

- Full design time capabilities

[◀ Back](#)

## **Data DropDown Methods**

[AddItem](#)

[AddItemBookmark](#)

[AddItemRowIndex](#)

[ColContaining](#)

[Columns](#)

[DoClick](#)

[Drag](#)

[GetBookmark](#)

[Groups](#)

[GrpContaining](#)

[MoveFirst](#)

[MoveLast](#)

[MoveNext](#)

[MovePrevious](#)

[MoveRecords](#)

[ReBind](#)

[Refresh](#)

[RemoveAll](#)

[RemoveItem](#)

[Reset](#)

[RowBookmark](#)

[RowContaining](#)

[RowTop](#)

[Scroll](#)

[SelBookmarks](#)

[StyleSets](#)

[◀ Back](#)

## **Data DropDown Objects**

[Bookmark](#)

[Column](#)

[Font](#)

[Group](#)

[HeadFont](#)

[ssRowBuffer](#)

[StyleSet](#)

[◀ Back](#)

## **Data DropDown Properties**

[\(About\)](#)

[\(Custom\)](#)

[BackColor](#)

[BackColorEven](#)

[BackColorOdd](#)

[BevelColorFace](#)

[BevelColorFrame](#)

[BevelColorHighlight](#)

[BevelColorScheme](#)

[BevelColorShadow](#)

[Bookmark](#)

[BorderStyle](#)

[CheckBox3D](#)

[Cols](#)

[ColumnHeaders](#)

[DataFieldList](#)

[DataFieldToDisplay](#)

[DataMode](#)

[DataSource](#)

[DefColWidth](#)

[DividerStyle](#)

[DividerType](#)

[DragIcon](#)

[DragMode](#)

[DroppedDown](#)

[Enabled](#)

[FieldDelimiter](#)

[FieldSeperator](#)

[FirstRow](#)

[Font](#)

[Font3D](#)

[ForeColor](#)

[ForeColorEven](#)

[ForeColorOdd](#)

[GroupHeaders](#)

[GroupHeadlines](#)

[HeadFont](#)

[HeadFont3D](#)

[Headlines](#)

[HeadStyleSet](#)


Height  
HelpContextID  
hWnd  
Index  
Left  
LeftCol  
LeftGrp  
LevelCount  
ListAutoPosition  
ListAutoValidate  
ListWidth  
ListWidthAutoSize  
MaxDropDownItems  
MinDropDownItems  
Name  
Redraw  
Row  
RowHeight  
Rows  
RowSelectionStyle  
Scrollbars  
StyleSet  
TabIndex  
Tag  
TagVariant  
Top  
UseExactRowCount  
Visible  
VisibleCols  
VisibleGrps  
VisibleRows  
WhatsThisHelpID  
Width

## Data DropDown: Exercise 1

This section guides you through the creation of a sample program using the Data DropDown control. For a complete description of this control, refer to the [Data DropDown control](#).

For the exercises in this section, it is assumed that you have already launched your development application (Visual Basic), and that the control has been added to your toolbox. For more information on how to do this, refer to [Using Data Widgets](#).

In this exercise, you will create a simple program that makes use of the Data DropDown control.

1. Place a Visual Basic data control on the form.
2. Set the **Visible** property to **False**. This hides the data control when your program runs.
3. Set the **DatabaseName** property to BIBLIO.MDB. Be sure to qualify the filename with the path of where the file is located, if needed.
4. Set the **RecordSource** property to 'Titles'
5. Place a **SSDBGrid** control directly on the form by double clicking on the  tool in the Visual Basic toolbox. Resize the grid to a size that is suitable for your form.
6. For the SSDBGrid control, set the **DataSource** property to *Data1*. This points the Data Grid to the data control you created in Step 1.
7. For the SSDBGrid control, set the **AllowAddNew** and **AllowDelete** properties to **True**.
8. Place a second Visual Basic data control on the form.
9. For the data control, set the **Visible** property to **False**.
10. For the data control, set the **DatabaseName** property to BIBLIO.MDB. Be sure to qualify the filename with the path of where the file is located, if needed.
11. For the data control, set the **RecordSource** property to 'Publishers'
12. Place a Data DropDown control on the form. The location of the Data DropDown is unimportant since it is invisible at runtime.
13. For the SSDBDropDown control, set the **DataSource** property of the Data DropDown to the second data control.
14. For the SSDBDropDown control, set the **DataFieldList** property of the Data DropDown to 'PubID'.
15. Link the Data DropDown to the Data Grid by adding the following code in the **InitColumnProps** procedure of the Data Grid:

```
SSDBGrid1.Columns(3).DropDownWnd = SSDBDropDown1.hWnd
```

Try running your application at this point. Click on the PubID column, and it will drop down the 'Publisher's table. Try changing the field value by selecting another record from the list.

By default, all fields in the table display. You can selectively hide fields by adding code in the **DropDown** event of the Data DropDown.

[◀ Back](#)

## **Data Grid Collections**

[Columns](#)

[Groups](#)

[SelBookmarks](#)

[StyleSets](#)



## Data Grid Control

[Properties](#)

[Events](#)

[FileNames](#)

[Methods](#)

[ObjectType](#)

[Collections](#)

[Objects](#)

### [Anatomy of a Data Grid](#)

The Data Grid custom control is an editable grid that can be used to display and edit data. In just a few steps, you can have a fully functional program that allows users to view, edit, add, and delete rows in a database without a single line of code! The Data Grid can operate in bound, unbound, or AddItem mode. When working in bound mode, the Data Grid communicates with the host environment's data control, which allows your grid to interact with any database the data control is capable of using.

In AddItem mode, the Data Grid can be used as a multi-column list box, in which case, it is not linked to a database. Since the Data Grid uses virtual data management techniques, meaning it can handle any amount of data without using up all of Windows memory, you can use it to handle large lists of data. When being used in add item mode, the Data Grid stores all data in memory, which is in contrast to bound and unbound modes where only the amount of data needed to display is kept in memory.

The Data Grid is fully-customizable and can contain multiple groups and columns with the ability to specify attributes such as colors, fonts, and user permissions to individual columns and groups.

The SSDBGrid control is zero-based, which means that numbering for all rows, columns, levels, etc. start at 0. For example, the command ?SSDBGrid1.Columns(1).Caption returns the caption of the **second** column in the grid.

When using the grid in bound mode, by default, each column represents a field in the database with each column header being named after the respective database field.

**Note** In Data Widgets 1.0, you could not see any groups, columns or special attributes when in design mode. Version 2.0 now allows you to view the grid at design time the way it will appear at runtime.

[A note about bookmarks](#)

[Adding a Bound Data Grid to your application](#)

[Adding an Unbound Data Grid to your application](#)

[Adding an AddItem Grid to Your Application](#)

[Grid Editor](#)

[Keyboard Interface](#)

[About SelBookmarks](#)

[Using the Data Grid as a List Box](#)

[Using the Cell Button Feature of the Data Grid](#)

[Using a Data DropDown in a Data Grid Column](#)

[Binding to a Data Control Across Forms](#)

[Updating Rows from a Modal Form](#)

[How the Data Grid handles data validation and error checking](#)

[How the DataGrid handles null values](#)



[◀ Back](#)

## **Data Grid Events**

[AfterColUpdate](#)

[AfterDelete](#)

[AfterInsert](#)

[AfterPosChanged](#)

[AfterUpdate](#)

[BeforeColUpdate](#)

[BeforeDelete](#)

[BeforeInsert](#)

[BeforeRowColChange](#)

[BeforeUpdate](#)

[BtnClick](#)

[Change](#)

[Click](#)

[ColMove](#)

[ColResize](#)

[ColSwap](#)

[ComboCloseUp](#)

[ComboDropDown](#)

[DbClick](#)

[DragDrop](#)

[DragOver](#)

[Error](#)

[GotFocus](#)

[GrpHeadClick](#)

[GrpMove](#)

[GrpResize](#)

[GrpSwap](#)

[HeadClick](#)

[InitColumnProps](#)

[KeyDown](#)

[KeyPress](#)

[KeyUp](#)

[LostFocus](#)

[MouseDown](#)

[MouseMove](#)

[MouseUp](#)

[RowColChange](#)

[RowLoaded](#)

[RowResize](#)

[Scroll](#)

ScrollAfter

SelChange

SplitterMove

UnboundAddData

UnboundDeleteRow


UnboundPositionData

UnboundReadData

UnboundWriteData

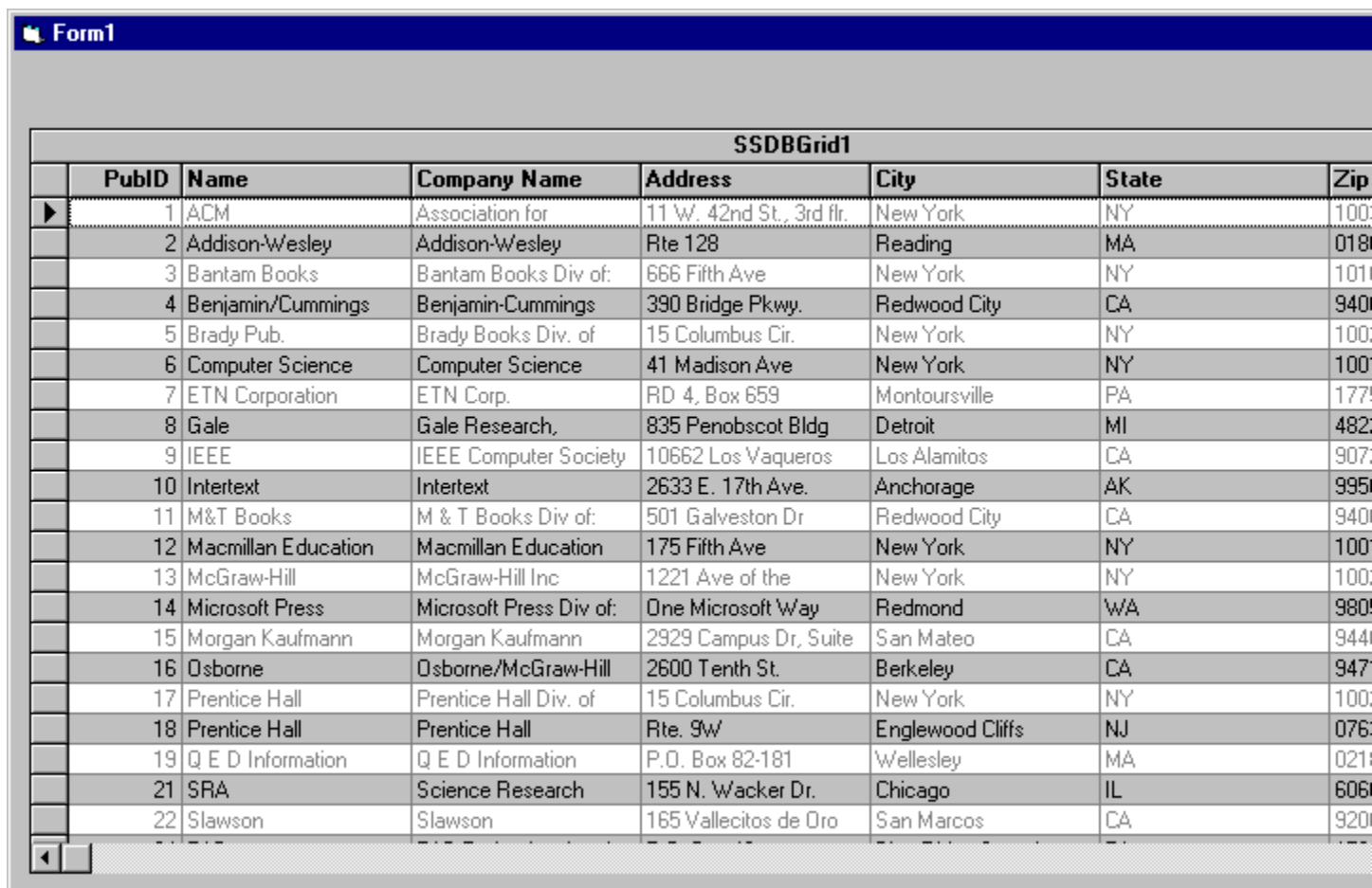
UpdateError

## Data Grid Exercise 1 - Part I: Adding the grid

1. Place a Visual Basic data control on the form.
2. Set the **Visible** property to **False**. This hides the data control when your program runs.
3. Set the **DatabaseName** property to BIBLIO.MDB. Be sure to qualify the filename with the path of where the file is located, if needed.
4. Set the **RecordSource** property to 'Publishers'
5. Place a SSDBGrid control directly on the form by double clicking on the  tool in the Visual Basic toolbox. Resize the grid to a size that is suitable for your form.
6. Set the **DataSource** property to *Data1*. This points the Data Grid to the data control you created in Step 1.
7. Set the **AllowAddNew** and **AllowDelete** properties to **True**.

Believe it or not, you just created a fully functional database grid in seven steps! You can view and edit the entire table, as well as add new rows and delete existing rows.

To see for yourself, try running the application at this point (select *Start* from the **Run** menu of Visual Basic). The results should look something like:



The screenshot shows a window titled "Form1" containing an SSDBGrid control. The grid displays a table with the following data:

SSDBGrid1							
	PubID	Name	Company Name	Address	City	State	Zip
▶	1	ACM	Association for	11 W. 42nd St., 3rd flr.	New York	NY	100
	2	Addison-Wesley	Addison-Wesley	Rte 128	Reading	MA	018
	3	Bantam Books	Bantam Books Div of:	666 Fifth Ave	New York	NY	101
	4	Benjamin/Cummings	Benjamin-Cummings	390 Bridge Pkwy.	Redwood City	CA	940
	5	Brady Pub.	Brady Books Div. of	15 Columbus Cir.	New York	NY	100
	6	Computer Science	Computer Science	41 Madison Ave	New York	NY	100
	7	ETN Corporation	ETN Corp.	RD 4, Box 659	Montoursville	PA	177
	8	Gale	Gale Research.	835 Penobscot Bldg	Detroit	MI	482
	9	IEEE	IEEE Computer Society	10662 Los Vaqueros	Los Alamitos	CA	907
	10	Intertext	Intertext	2633 E. 17th Ave.	Anchorage	AK	995
	11	M&T Books	M & T Books Div of:	501 Galveston Dr	Redwood City	CA	940
	12	Macmillan Education	Macmillan Education	175 Fifth Ave	New York	NY	100
	13	McGraw-Hill	McGraw-Hill Inc	1221 Ave of the	New York	NY	100
	14	Microsoft Press	Microsoft Press Div of:	One Microsoft Way	Redmond	WA	980
	15	Morgan Kaufmann	Morgan Kaufmann	2929 Campus Dr, Suite	San Mateo	CA	944
	16	Osborne	Osborne/McGraw-Hill	2600 Tenth St.	Berkeley	CA	947
	17	Prentice Hall	Prentice Hall Div. of	15 Columbus Cir.	New York	NY	100
	18	Prentice Hall	Prentice Hall	Rte. 9W	Englewood Cliffs	NJ	076
	19	Q E D Information	Q E D Information	P.O. Box 82-181	Wellesley	MA	021
	21	SRA	Science Research	155 N. Wacker Dr.	Chicago	IL	606
	22	Slawson	Slawson	165 Vallecitos de Oro	San Marcos	CA	920

## Data Grid Exercise 1 - Part II: Creating groups and columns

While the grid displays all your data, wouldn't it be nice to spice it up a little? In this next part, we're going to make use of a powerful tool called the Grid Editor. With the Grid Editor, we'll be able to create groups and columns, as well as easily specify a variety of display attributes.

With the application we created in Part I open, let's do the following:

- In Design mode, select the grid and then select "(Custom)" from the *Properties* list. The Grid Editor appears.

The first thing we want to do is divide the various fields into organizational groups:

1. Select the "Groups" tab. The Groups tab allows us to create and delete group definitions.
2. Click the **Add** button and type "Company Information"
3. Click the **Add** button and type "Address Information"
4. Click the **Add** button and type "Other"

You have just created three groups. Now, it's time to add some fields (herein referred to as "Columns") to the groups.

1. Select the "Columns" tab. The Columns tab allows us to create and delete column definitions.
2. Click on the group header labeled "Company Information".
3. Click the **Fields** button. The Fields button allows us to automatically create columns from a bound database.
4. Select the fields "Name" and "Company Name" from the *Fields Selection* list.
5. Click the **OK** button. You have just added these two fields to the "Company Information" group.
6. Click on the group header labeled "Address Information".
7. Click the **Fields** button.
8. Select the fields "Address", "City", "State", and "Zip" from the *Fields Selection* list.
9. Click the **OK** button. You have just added these four fields to the "Address Information" group.
10. Click on the group header labeled "Other".
11. Select the fields "Telephone", "Fax", and "Comments" from the *Fields Selection* list.
12. Click the **OK** button. You have just added these four fields to the "Other" group.

You have just created a grid layout making use of groups and columns. Resize the groups to your liking by clicking on the right edge of the group headers and dragging them to either the left for smaller, or right for larger. You can do the same for the columns by clicking on the column headers and dragging. You'll notice that we left two fields out of our grid, "PubID" and "Comments". The Grid Editor allows you to selectively use fields in your grid.

Once you have specified your layout, it's a good idea to actually *Apply* it to the grid by clicking the **Apply** button. This updates the SSDBGrid control with the layout you just

designed. You'll then want to close the Grid Editor for now by clicking the **OK** button.

**Note** You can apply your changes *and* close the Grid Editor simply by clicking the **OK** button. If your changes have not yet been applied, you will be prompted to apply them prior to closing.

If you want, try running the application at this point (select *Start* from the **Run** menu of Visual Basic). The results should look something like:

Form1

SSDBGrid1								
Company Information			Address Information				Other	
Name	Company Name	Address	City	State	Zip	Telephone	Fax	
ACM	Association for Computing	11 W. 42nd St.,	New York	NY	10036	212-869-7440		
Addison-Wesley	Addison-Wesley Publishing Co	Rte 128	Reading	MA	01867	617-944-3700	617-96	
Bantam Books	Bantam Books Div of: Bantam	666 Fifth Ave	New York	NY	10103	800-223-6834	212-76	
Benjamin/Cummings	Benjamin-Cummings Publishing	390 Bridge Pkwy.	Redwood City	CA	94065	800-950-2665	415-59	
Brady Pub.	Brady Books Div. of Prentice Hall	15 Columbus Cir.	New York	NY	10023	212-373-8093	212-37	
Computer Science Press	Computer Science Press Inc	41 Madison Ave	New York	NY	10010	212-576-9400	212-68	
ETN Corporation	ETN Corp.	RD 4, Box 659	Montoursville	PA	17754-94	717-435-2202	717-43	
Gale	Gale Research, Incorporated	835 Penobscot	Detroit	MI	48226-40	313-961-2242	313-96	
IEEE	IEEE Computer Society Press	10662 Los	Los Alamitos	CA	90720	800-272-6657	714-82	
Intertext	Intertext Publications/Multiscience	2633 E. 17th Ave.	Anchorage	AK	99508			
M&T Books	M & T Books Div of: M&T	501 Galveston Dr	Redwood City	CA	94063-47	800-533-4372	415-36	
Macmillan Education	Macmillan Education Ltd	175 Fifth Ave	New York	NY	10010	212-460-1500		
McGraw-Hill	McGraw-Hill Inc	1221 Ave of the	New York	NY	10020	212-512-2000		
Microsoft Press	Microsoft Press Div of: Microsoft	One Microsoft Way	Redmond	WA	98052-63	800-MSPRESS	206-88	
Morgan Kaufmann	Morgan Kaufmann Publishers Inc.	2929 Campus Dr.	San Mateo	CA	94403	415-578-9911	415-57	
Osborne	Osborne/McGraw-Hill Div. of	2600 Tenth St.	Berkeley	CA	94710	800-227-0900		
Prentice Hall	Prentice Hall Div. of Simon &	15 Columbus Cir.	New York	NY	10023	800-922-0579		
Prentice Hall	Prentice Hall International,	Rte. 9W	Englewood	NJ	07632	201-592-2000		
Q E D Information	Q E D Information Sciences,	P.O. Box 82-181	Wellesley	MA	02181	800-343-4848	617-23	
SRA	Science Research Associates	155 N. Wacker Dr.	Chicago	IL	60606	800-621-0476	312-98	

## Data Grid Exercise 1 - Part III: Customizing the Data Grid

The Data Grid is quite versatile when it comes to customizing both look and functionality. The Grid Editor allows you to work with a number of properties at design time. In this next section, we're going to customize our grid so that you can begin to see the wide-range of possibilities the grid offers.

At this point, we want to go back into the grid editor so that we can customize our grid. If you don't remember, we can launch the grid editor by selecting the grid control then select "(Custom)" from the *Properties* list.

Changes made in the Grid Editor will not take affect in the real grid until we click the **Apply** button in the Grid Editor.

Let's begin to customize our grid:

### Customizing General tab options

The **General** tab is a tree-structure representing the various properties that can be set for the Data Grid. The General tab is the first tab to appear when you launch the Grid Editor. When you select a property for modification, options for that property appear to the right. The two exceptions to this are the (Add Items...) and StyleSets options, which are both fully explained in 'Chapter 11 - The Data Grid Control'.

Set the following properties as shown:

<b>Property</b>	<b>Value</b>	<b>What It Does</b>
<b>Caption</b>	"Publisher Database"	Specifies the caption title for the Data Grid
<b>CaptionAlignment</b>	"0 - Left Justify"	Left justifies the caption
<b>AllowAddNew</b>	True	Allows users to add new records to the Data Grid
<b>AllowDelete</b>	True	Allows users to delete records from the Data Grid
<b>Font</b>	"Arial" for Name 8 for Point Size	Specifies the font name and size to be used for the grid text.
<b>HeadFont3D</b>	"Inset w/light shading"	Gives the text of your grid headers a 3D appearance.
<b>HeadFont</b>	"Arial" for Name 12 for Point Size	Specifies the font name and size to be used for the grid headers, including caption.
<b>AllowColumnMoving</b>	"2 - Anywhere"	Allows users to move columns anywhere on the grid
<b>AllowColumnSwapping</b>	"2 - Anywhere"	Allows users to swap columns anywhere on

<b>DividerType</b>	"Horizontal"	the grid Determines what type of row divider is used.
<b>SelectByCell</b>	True	Allows the selection of an entire row if the user clicks on a cell.
<b>GroupHeadLines</b>	2	Allows the Group Headers to occupy two rows.
<b>LevelCount</b>	2	Allows each record to occupy two rows. You will need to decide what columns you want on each level.

## Customizing Group options

1. Select the "Groups" tab.
2. Select the group "Other" from the **Name** list.
3. Set the **AllowSizing** property to False. This prevents users from changing the width of the "Other" group.
4. Select the group "Company Information" from the **Name** list.
5. Set the **HeadBackColor** property to the color that you want to use. The color you choose can either be from the palette shown, a custom color you define by clicking on the **Custom Color** list, or a pre-defined system color from the **System Color** list that corresponds to your Windows color scheme.
6. Repeat Step 5 for as many groups as you want to define colors for.
7. Move the "Company Name", "City", "State", and "Zip" columns so that they appear on the second level.

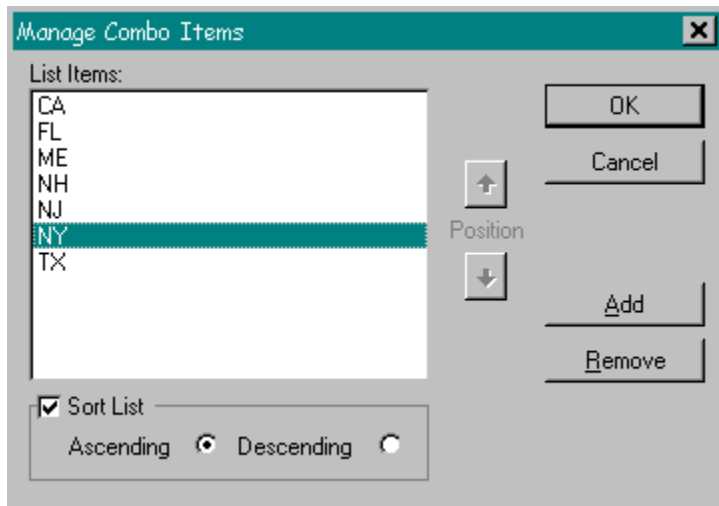
To move a column from one level to another, select the column header and drag it to the appropriate level. The results will look something like:

SSDBGrid1					
Company Information		Address Information			Other
Name		Address			Telephone
Company Name		City	State	Zip	Fax
This is an even line					
This is an odd line					

## Customizing Column options

1. Select the "Columns" tab.
2. Select the column "Name" from the **Name** list.
3. Set the **Case** property to '2 - UPPER'. This causes all fields in the "Name" column to appear in uppercase.

4. Select the column "State" from the **Name** list.
5. Set the **Style** property to '3 - Combo Box'. This causes the State field to appear as a combo box, allowing the user to choose from a list of entries. When you select **Combo Box**, the **Setup** button appears, allowing you to modify the contents of the combo box.
6. Click the **Setup** button. The Manage Combo Items dialog appears:



7. Add the names of states listed in the graphic above by clicking the **Add** button and then typing the name in the *Add List Item* edit box.
8. Select **Sorted** by clicking in the check box. This will sort your entries in ascending or descending order based on your preference. You can manually sort the list by selecting fields individually and clicking the **Up** or **Down** buttons.
9. Click the **OK** button.

You've just completed modifying the grid layout! Remember, in order for the changes we specify in the Grid Editor to take affect on the actual grid, you must click the **Apply** button to activate the changes. We're done with the Grid Editor, so you can click the **OK** button to go back to the form.



## Data Grid Exercise 1 - Part IV: Running the application

You can now run your application to see how the changes have affected your grid. Some items to note about the grid are:

- Notice how the Company Name field displays all entries in uppercase.
- Try clicking on the State column. A combo box should appear listing the states you entered.
- Try resizing groups and columns by selecting the right side of the column or group header and dragging left or right. Try doing this for the "Other" group. Remember that you disabled the resizing for this group.

The Grid Editor can be run again at anytime in the future, that is, you can make further changes to your grid whenever you're in design mode.

The exercise just completed is just a taste of what's possible with the Data Grid. The best way to learn about the grid is to experiment with different settings. The Grid Editor provides context-sensitive help throughout should you have any questions about a specific property. You should also refer to Chapter 11 for an in-depth view of the Data Grid control.

## Data Grid Features

The Data Grid is a fully editable bound grid that allows you to edit an entire record set, regardless of size, on screen without writing any code.

The following is a list of the features found in the Data Grid control:

- Functionally and visually consistent with data grids in Microsoft Access and Visual Basic 4.0

- Support for movable groups and columns

- Optional dropdowns in headings allow users to select from a list of available fields and/or groups at runtime

- Additional cell types include checkbox, button, label, and combo box

- Multiline row formats

- Pictures and text in cells and headings

- Use of fonts and colors by column, row, and cell

- Drag and Drop of cells

- Supports multiple data modes including bound, unbound, and AddItem.

- AddItem at design time

- Supports Sheridan Style Sets

[◀ Back](#)

## **Data Grid Methods**

[ActiveCell](#)

[AddItem](#)

[AddItemBookmark](#)

[AddItemRowIndex](#)

[AddNew](#)

[CancelUpdate](#)

[ColContaining](#)

[ColPosition](#)

[Columns](#)

[DeleteSelected](#)

[DoClick](#)

[Drag](#)

[GetBookmark](#)

[Groups](#)

[GrpContaining](#)

[GrpPosition](#)

[IsAddrow](#)

[IsItemInList](#)

[MoveFirst](#)

[MoveLast](#)

[MoveNext](#)

[MovePrevious](#)

[MoveRecords](#)

[ReBind](#)

[Refresh](#)

[RemoveAll](#)

[RemoveItem](#)

[Reset](#)

[RowBookmark](#)

[RowContaining](#)

[RowTop](#)

[Scroll](#)

[SelBookmarks](#)

[StyleSets](#)

[Update](#)

[WhereIs](#)

[◀ Back](#)

## **Data Grid Objects**

[ActiveCell](#)

[Column](#)

[Font](#)

[Group](#)

[HeadFont](#)

[ssRowBuffer](#)

[StyleSet](#)

[◀ Back](#)

## **Data Grid Properties**

[\(About\)](#)

[\(Custom\)](#)

[ActiveRowStyleSet](#)

[Align](#)

[AllowAddNew](#)

[AllowColumnMoving](#)

[AllowColumnShrinking](#)

[AllowColumnSizing](#)

[AllowColumnSwapping](#)

[AllowDelete](#)

[AllowDragDrop](#)

[AllowGroupMoving](#)

[AllowGroupSizing](#)

[AllowGroupSwapping](#)

[AllowGroupShrinking](#)

[AllowRowSizing](#)

[AllowUpdate](#)

[BackColor](#)

[BackColorEven](#)

[BackColorOdd](#)

[BalloonHelp](#)

[BevelColorFace](#)

[BevelColorFrame](#)

[BevelColorHighlight](#)

[BevelColorScheme](#)

[BevelColorShadow](#)

[Bookmark](#)

[BorderStyle](#)

[Caption](#)

[CaptionAlignment](#)

[CellNavigation](#)

[CheckBox3D](#)

[Col](#)

[ColChanged](#)

[Cols](#)

[ColumnHeaders](#)

[ComboDroppedDown](#)

[DataChanged](#)

[DataMode](#)

[DataSource](#)

DefColWidth  
DividerStyle  
DividerType  
DragIcon  
DragMode  
Enabled  
FieldDelimiter  
FieldSeparator  
FirstRow  
Font  
Font3D  
ForeColor  
ForeColorEven  
ForeColorOdd  
GroupHeaders  
GroupHeadlines  
Grp  
HeadFont  
HeadFont3D  
Headlines  
HeadStyleSet  
Height  
HelpContextID  
hWnd  
hWndEdit  
Index  
Left  
LeftCol  
LeftGrp  
LevelCount  
MaxSelectedRows  
MultiLine  
Name  
Negotiate  
PictureButton  
PictureComboButton  
PictureRecordSelectors  
RecordSelectors  
Redraw  
ResizeHeight  
ResizeWidth  
Row

[RowChanged](#)  
[RowHeight](#)  
[RowNavigation](#)  
[Rows](#)  
[RowSelectionStyle](#)  
[Scrollbars](#)  
[SelectByCell](#)  
[SelectTypeCol](#)  
[SelectTypeRow](#)  
[SplitterPos](#)  
[SplitterVisible](#)  
[StyleSet](#)  
[TabIndex](#)  
[TabNavigation](#)  
[TabStop](#)  
[Tag](#)  
[TagVariant](#)  
[Top](#)  
[UseDefaults](#)  
[UseExactRowCount](#)  
[Visible](#)  
[VisibleCols](#)  
[VisibleGrps](#)  
[VisibleRows](#)  
[WhatsThisHelpID](#)  
[Width](#)

**See Also**

[How the DataGrid handles null values](#)



## **Data Grid: Exercise 1 (Bound Mode)**

This section guides you through the creation of some sample programs using the Data Grid control. For a complete description of this control, refer to the [Data Grid Control](#).

For this exercise, it is assumed that you have already launched your development application (Visual Basic), and that the control has been added to your toolbox. For more information on how to do this, refer to [Using Data Widgets](#).

In this exercise, you will create an application that makes use of the Data Grid control. The database used will be the BIBLIO.MDB that ships with Visual Basic and is located in the Visual Basic home directory. If you are using an environment other than Visual Basic, and do not have the BIBLIO.MDB database, consult [BIBLIO File Structure](#) for details on the file layout.

**Part I: Adding the grid**

**Part II: Creating groups and columns**


**Part III: Customizing the Data Grid**

**Part IV: Running the application**

## Data Grid: Exercise 2 (Unbound Mode)

In this exercise, you will create a fully functional unbound Data Grid control. The database used will be the UNBOUND.MDB that resides in the \SAMPLES\CHAP5 directory of Data Widgets 2.0. The reason this is an unbound data grid is because we are not using the Visual Basic data control, and are using our own routines to deal with data.

**Note** This exercise makes use of the Microsoft DAO 2.5 Object Library by using Visual Basic commands such as `OpenDatabase` and `OpenRecordset`. If you do not have this library available, VB may generate a "User-defined type not defined" message when running this application. If you encounter this situation, simply enable this library through the "References..." dialog under Visual Basic's *Tools* menu.

1. Place a SSDBGrid control directly on the form by double clicking on the  tool in the Visual Basic toolbox. Resize the grid to a size that is suitable for your form.
2. Set the **DataMode** property to '1 - Unbound'.
3. Set the **AllowAddNew** and **AllowDelete** properties to 'True'
4. Add the following code to the **(declarations)** section of your form:  

```
Dim db As Database
Dim rs As Recordset
Dim ct As Integer
```

```
' DB represents the database to be used
' RS represents the recordset to be used
' CT represents the count of fields in the table
```

5. Add the following code to the **Form\_Load** procedure:

```
Dim i As Integer
Set db = OpenDatabase("unbound.mdb")
Set rs = db.OpenRecordset("Titles")
ct = rs.Fields.Count

SSDBGrid1.Columns.RemoveAll

For i = 0 To ct - 1
    SSDBGrid1.Columns.Add i
    SSDBGrid1.Columns(i).Visible = True
    SSDBGrid1.Columns(i).Caption = rs.Fields(i).Name
Next i
```

In this section, the database is assigned, a table selected, and the number of fields in the table is given to CT. The RemoveAll is issued to ensure that there are no existing columns when the grid goes to add columns from the database. In the `For i = 0 To ct - 1` loop, columns are created based on the number of fields in the database, and headers are assigned based on the field name.

6. Add the following code to the **SSDBGrid1\_UnboundReadData** event:

```
Dim r, i, j As Integer

If rs.RecordCount = 0 Then Exit Sub
If IsNull (StartLocation) Then
    If ReadPriorRows Then
```

```

        rs.MoveLast
    Else
        rs.MoveFirst
    End If

Else
    rs.Bookmark = StartLocation
    If ReadPriorRows Then
        rs.MovePrevious
    Else
        rs.MoveNext
    End If

End If

For i = 0 to Rowbuf.RowCount - 1
    if rs.BOF or rs.EOF Then Exit For

    For j = 0 to ct - 1
        Rowbuf.Value(i,j) = rs(j)
    Next j

    Rowbuf.Bookmark(i) = rs.Bookmark

    If ReadPriorRows Then
        rs.MovePrevious
    Else
        rs.MoveNext
    EndIf

    r = r + 1

Next i

Rowbuf.RowCount = r

```

The `UnboundReadData` event reads data into the grid. In this section, the program first checks to see if it's dealing with an empty recordset so it can exit before attempting the read. It then looks to see if it's dealing with an empty grid or not by checking `StartLocation`. If the grid is empty, then the code moves to the first or last record based on the `ReadPriorRows` value (which indicates if scrolling should occur forwards or backwards). If the grid is not empty, the database bookmark is set to the start location and we move backwards or forwards based on `ReadPriorRows`.

We then read in the data, only stopping when we reach the beginning or end the file or when we reach `RowCount - 1`. As we add each row, we keep track of a counter `r`, which we set `RowBuf.RowCount` to when we finish.

7. Add the following code to the **SSDBGrid1\_UnboundAddData** event:

```

Dim i As Integer

rs.AddNew

For i = 0 to (ct-1)
    If Not IsNull (Rowbuf.Value (0,i)) Then
        rs(i) = Rowbuf.Value(0,i)
    End If
Next i

```

```

    End If
Next i

rs.Update
rs.MoveLast
NewRowBookmark = rs.Bookmark

```

The **UnboundAddData** event adds a new row. In this section, the database is prepared for a row addition, and the field values are passed from the `ssRowBuffer` to the database for each field. The database is then updated. Finally, we move to the last row in the database, and point the `NewRowBookmark`.

8. Add the following code to the **SSDBGrid1\_UnboundDeleteRow** event:

```

rs.Bookmark = Bookmark
rs.Delete

```

The **UnboundDeleteRow** event deletes a row. In this section, the database is pointed to the row being deleted, and then actually deletes it.

9. Add the following code to the **SSDBGrid1\_UnboundWriteData** event:

```

Dim I As Integer

rs.Bookmark = WriteLocation

rs.Edit

For i = 0 To (ct-1)
    If Not IsNull(RowBuf.Value(0,i)) Then
        rs(i) = RowBuf.Value(0,i)
    End If
Next i

rs.Update


```

The **UnboundWriteData** event writes a row that has been changed. In this section, the database is pointed to the row being written, and then put into edit mode. The data is then passed from the grid to the database for each field, and then issued an update command.

10. Run your application.

## Data Grid: Exercise 3 (AddItem Mode)

In this exercise, you will create a fully functional AddItem Data Grid control.

1. Place a SSDBGrid control directly on the form by double clicking on the  tool in the Visual Basic toolbox. Resize the grid to a size that is suitable for your form.
2. Set the **DataMode** property to '2 - AddItem'.
3. Set the **Cols** property to 3. This tells the grid that we will use three columns.
4. Set the **FieldDelimiter** property to ! (exclamation mark). This property determines the character that represents the start and end of a field.
5. Set the **FieldSeparator** property to , (comma). This property determines the character that represents a separation between fields.
6. Add the following code to the **SSDBGrid1\_InitColumnProps** event:

```
Dim I As Integer
For I = 0 to 20
    SSDBGrid1.AddItem "!Hello!,!There!,!World!"
Next I
```

Run your program. You'll see that it functions just like an Unbound or Bound grid.

Select a Data Widget control:

Data Grid

Data Combo

Data DropDown

DataOptionSet

Enhanced Data

Data Command Button

## DataField Property

[See Also](#)

[Applies To](#)

### Description

Determines a field to bind to in the current database.

### Syntax

*object* . **DataField**[= *value*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Value</i>	A string expression that evaluates to the name of the field in the object specified in the <b>DataSource</b> property.

### Remarks

You must first specify a source in the **DataSource** property.

## **DataField Property Applies To**

Column object

SSDBCombo

SSDBCommand

SSDBData

SSDBOptSet



## **DataField Property See Also**

[DataSource](#)

## DataFieldList Property

[See Also](#)

[Applies To](#)

### Description

Returns or sets a value that determines the data field to be used for the dropdown portion of the control.

### Syntax

*object* . **DataFieldList**[= *string*]

Part	Description
------	-------------

---

<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
---------------	---

<i>string</i>	A string expression that evaluates to the name of one of the fields in the Recordset object specified by a Data control's <b>RecordSource</b> and <b>DatabaseName</b> properties.
---------------	---

### Remarks

The **DataField List** property determines which field will be used to return the data to the edit portion of the control when an item is selected from the list.

The **DataSourceList** property of the SSDBCombo control specifies a valid Data control name, and the **DataFieldList** property specifies a valid field name in the Recordset object created by the Data control.



**Important!** The **DataFieldList** property must be set in order for the combo to drop down. This is true in any mode of the control (bound, unbound or AddItem.) **If this property is not set, the combo will not drop down.**

## **DataFieldList Property Applies To**

SSDBCombo

SSDBDropDown

## **DataFieldList Property See Also**

[DataMode](#)

[DataSource](#)

[DataSourceList](#)

## DataFieldToDisplay Property

[See Also](#)

[Applies To](#)

### Description

Determines the field to display in the edit portion of the control.

### Syntax

*object* . **DataFieldToDisplay**[= *value*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>value</i>	A string expression that evaluates to the name of the field (in the object specified in the <b>DataSourceList</b> property) that will be displayed in the edit portion of the control.

### Remarks

DataFieldToDisplay is used when you wish to display one value to the user while storing a different value in the database field attached to the control. The classic example is a State field, where you want to display the name of the state, such as "Pennsylvania", but store only a code, such as "PA" in the database. **DataFieldToDisplay** determines the field that will be displayed, the **DataFieldList** property will determine the field that is stored in the database.

If **DataFieldToDisplay** is left blank, then the field displayed in the edit portion of the control is the same field stored in the database, as determined by **DataFieldList** and **DataField**.

For best performance in bound mode, the **DataFieldList** column (which determines the value to be stored) should be included in the dropdown. If you do not want the values of this column displayed to the user, you can hide it by setting its **Visible** property to False. If the **DataFieldList** column is not included in the grid's layout, references to the field will have to be resolved at run time, adversely affecting performance.

If the control is not in bound mode, the the field specified by **DataFieldList** must be included as a column in the displayed grid.

To support **DataFieldToDisplay** in unbound mode, there is a corresponding setting for the **ReadType** property of the ssRowBuffer object. By setting **ReadType** to 3, you can optimize the retrieval of data from an unbound source.

**Note** A DataDropDown with a value specified for **DataFieldToDisplay** can only service a single column in the DataGrid to which it is attached. If you have a DataDropDown that services multiple columns in a DataGrid, you will not be able to use **DataFieldToDisplay**. To use **DataFieldToDisplay**, you must have only one DataDropDown per column.

## **DataFieldToDisplay Property Applies To**

SSDBCombo

SSBBDropDown

## **DataFieldToDisplay Property See Also**

[DataField](#) property

[DataFieldList](#) property

[ReadType](#) property

## DataMode Property

Applies To

### Description

Sets/Returns the mode used by control for data access.

### Syntax

*object* . **DataMode**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the data access mode used, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
0	(Default) Bound mode
1	Unbound mode
2	AddItem mode

There are constants available for the settings of this property.

### Remarks

Bound mode indicates that control will receive information from a database.

Unbound mode indicates that control will receive the data from another source, such as an array or unstructured source.

AddItem mode indicates that control will receive data from the programmer.



## **DataMode Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

[◀ Back](#)

## **DataOptionSet Collections**

Buttons



## DataOptionSet Control

[Properties](#)

[Events](#)

[Methods](#)

[FileNames](#)

[ObjectType](#)

[Collections](#)

[Objects](#)

The DataOptionSet custom control allows you to use 3D options buttons that can be bound to a database field. Using the DataOptionSet control, you can easily incorporate option buttons for use in your database application. Instead of entering data which can lead to typographical errors, users may simply click on an option button to select a pre-defined value for a field. For example, using one DataOptionSet, you could have four option buttons to represent various credit-card payment methods.

If the value of the active field equals the value set for the control, the option button will be automatically clicked. If another DataOptionSet button matches a value in the field or there is no match, the button will automatically be clicked off.

One control can contain multiple option buttons, which can be added (up to 100 total) or deleted, as specified by you at either design or runtime. Appearance of the buttons are user-controllable through the use of layout properties. The DataOptionSet is zero based, meaning that numbering of buttons starts at zero.

The DataOptionSet makes use of [objects and collections](#).



[Keyboard Interface](#)

[Adding the DataOptionSet](#)

[Creating Buttons at Design Time](#)

[Creating Buttons at Runtime](#)

[Binding to a Data Control Across Forms](#)

[◀ Back](#)

## **DataOptionSet Events**

[Click](#)

[DbClick](#)

[DragDrop](#)

[DragOver](#)

[GotFocus](#)

[KeyDown](#)

[KeyPress](#)

[KeyUp](#)

[LostFocus](#)

[MouseDown](#)

[MouseMove](#)

[MouseUp](#)

## DataOptionSet Features

The DataOptionSet control allows the binding of data fields to option buttons for representation of field values. For example, if you were writing a point-of-sale system, you could have four option buttons, each representing various credit cards ("Visa", "MasterCard", "American Express", and "Discover"). Clicking on the appropriate option button automatically changes the value in the database and allows you to store the type of payment that was used.

The following is a list of the features for this control:

- Multiline captions

- One control creates unlimited option buttons that are bound to the same data field

- Saves Windows resources

- Automatic and manual row/column positioning

- Each individual button can have a caption with optional picture

- Custom color options

- 3D capability

[◀ Back](#)

## **DataOptionSet Methods**

[ButtonFromCaption](#)

[ButtonFromPos](#)

[Drag](#)

[Refresh](#)

[WhereIs](#)

[◀ Back](#)

## **DataOptionSet Objects**

Button

[◀ Back](#)

## **DataOptionSet Properties**

[\(About\)](#)

[\(Custom\)](#)

[Alignment](#)

[BackColor](#)

[BevelColorFace](#)

[BevelColorFrame](#)

[BevelColorHighlight](#)

[BevelColorScheme](#)

[BevelColorShadow](#)

[ButtonEnabled](#)

[ButtonVisible](#)

[Caption](#)

[ColOffset](#)

[Cols](#)

[ColWidth](#)

[DataField](#)

[DataSource](#)

[DragIcon](#)

[DragMode](#)

[Enabled](#)

[Font](#)

[Font3D](#)

[ForeColor](#)

[Height](#)

[HeightGap](#)

[HelpContextID](#)

[hWnd](#)

[Index](#)

[IndexSelected](#)

[MaintainBtnHeight](#)

[MinColWidth](#)

[MinHeight](#)

[MouseIcon](#)

[MousePointer](#)

[Name](#)

[NumberOfButtons](#)

[OptionValue](#)

[Picture](#)

[PictureAlignment](#)

[PictureMetaHeight](#)



PictureMetaWidth

RowOffset

TabIndex

TabStop

Tag

Top

Visible

WhatsThisHelpID

Width

WidthGap

WordWrap

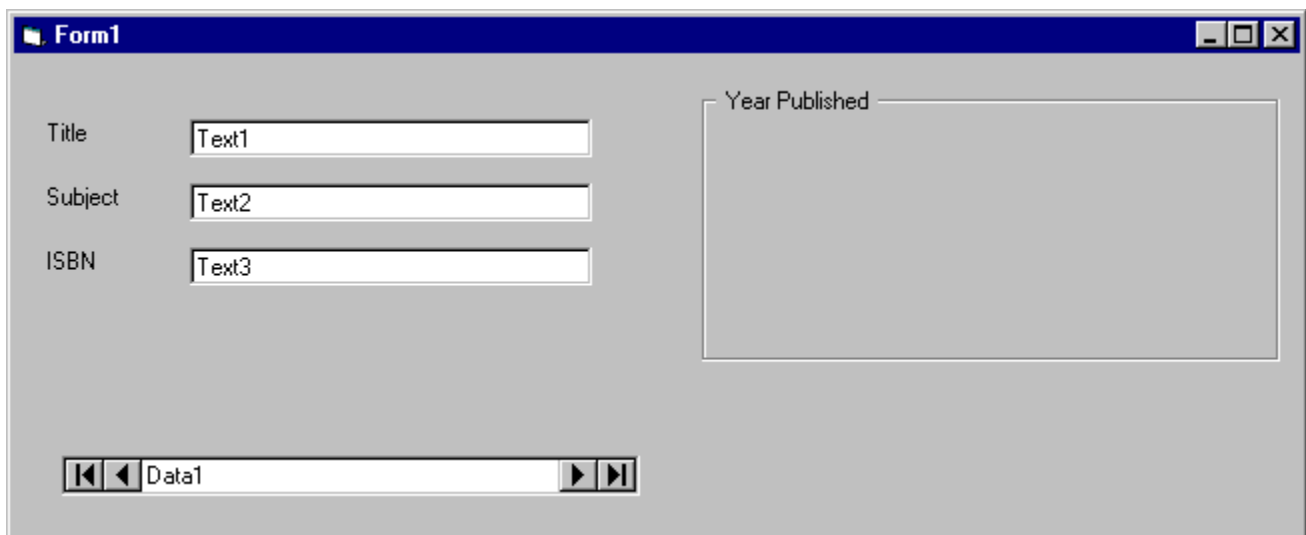
## DataOptionSet: Exercise 1


This section guides you through the creation of a sample program using the DataOptionSet control. For a complete description of this control, refer to the [DataOptionSet](#).

For this exercise, it is assumed that you have already launched your development application (Visual Basic), and that the control has been added to your toolbox. For more information on how to do this, refer to [Using Data Widgets](#).

In this exercise, you will create an application that makes use of the DataOptionSet control. The database used will be the BIBLIO.MDB that ships with Visual Basic and is located in the Visual Basic home directory. If you are using an environment other than Visual Basic, and do not have the BIBLIO.MDB database, consult [BIBLIO File Structure](#) for details on the file layout.

1. Place a standard data control on the form.
2. Set the **DatabaseName** property to BIBLIO.MDB and set the **RecordSource** property to 'Titles'.
3. Add three standard labels, three standard text boxes and a standard frame as shown below. Set the **DataSource** property to 'Data1' for all three text boxes. Set the **DataField** for the first text box to 'Title', set the second one to 'Subject', and the third to 'ISBN'.

The image shows a screenshot of a Visual Basic form titled "Form1". On the left side, there are three labels: "Title", "Subject", and "ISBN". Each label is followed by a text box containing "Text1", "Text2", and "Text3" respectively. To the right of these is a larger rectangular frame with the label "Year Published" at the top left. At the bottom of the form, there is a DataOptionSet control. It consists of a horizontal bar with the text "Data1" in the center, and navigation arrows (back, forward, first, last) on either side.

4. Add a DataOptionSet  control to the form, within the frame captioned 'Year Published'.
5. For the SSDBOptSet control, set the **DataSource** property to Data1 and the **DataField** property to 'Year Published'. This binds the control to the 'Year Published' field of the database specified in Data1.
6. Also for the SSDBOptSet control, set the **NumberOfButtons** property to 10 and the **Cols** property to 2. This creates ten option set buttons on the form, divided into two columns.
7. Set the **Caption** property to "1986". This changes the caption for the first button to 1986.
8. Set the **IndexSelected** property to 1. This changes the selected button from the first (0) to the second (1). The DataOptionSet is zero-based. When you first create a DataOptionSet, the first button is automatically selected.
9. Repeat Steps 7 and 8 until you have renamed all 10 buttons to look like:

Year Published

<input type="radio"/> 1986	<input type="radio"/> 1991
<input type="radio"/> 1987	<input type="radio"/> 1992
<input type="radio"/> 1988	<input type="radio"/> 1993
<input type="radio"/> 1989	<input checked="" type="radio"/> 1994
<input type="radio"/> 1990	<input type="radio"/> 1995

10. For each button, set the **OptionValue** property to match its caption. Remember, you must first set the **IndexSelected** property to specify the button you want to work with. The **OptionValue** property compares against the value in the database.
11. Run your application.

Try moving through some of the records using the data control. Notice how the option buttons change as you move. You'll notice that some records cause the DataOptionSet to not select any button, this is because the date does not match any of the option values specified. Remember, you didn't need to write a single line of code! However, it is possible to accomplish what we just did through code. The following procedure can be used in place of Steps 6 through 10:

```
Private Sub Form_Load()

    Dim iC as Integer
    Dim iYear as Integer

    SSDBOptSet1.NumberOfButtons = 10
    SSDBOptSet1.Cols = 2

    For iC = 0 to 9

        iYear = 1986 + iC
        SSDBOptSet1.Buttons(iC).Caption = Str(iYear)
        SSDBOptSet1.Buttons(iC).OptionValue = iYear

    Next iC

End Sub
```

You can begin to see how easy it is to work with DataOptionSet buttons through code.

## DataSourceList Property

[See Also](#)

[Applies To](#)

### Description

Sets a value that specifies the Data control through which the list portion of the SSDBCombo control is bound to a database. Not available at run time.

### Remarks

To bind the list portion of an SSDBCombo control to a field in a database at run time, you must specify the name of a Data control in the **DataSourceList** property at design time using the Properties window.

The **DataSourceList** property of the SSDBCombo control specifies a valid Data control name, and the **DataFieldList** property specifies a valid field name in the Recordset object created by the Data control.

To complete the connection with a field in the Recordset managed by the Data control, you must also provide the name of a Field object in the **DataFieldList** property. Unlike the **DataFieldList** property, the **DataSourceList** property setting isn't available at run time.

## **DataSourceList Property Applies To**

SSDBCombo

## **DataSourceList Property See Also**

[DataFieldList](#)

[DataMode](#)

[DataSource](#)

## Data Type Property

Applies To

### Description

Returns the column's underlying data type.

### Syntax

*object* . **Data Type**[= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the underlying data type, as described in Settings.

### Settings

Value	Description
2	Integer
3	Long
4	Single
5	Double
6	Currency
7	Date
8	(Default) Text
11	Boolean
17	Byte

There are constants available for the settings of this property.

### Remarks

This property can be set only if working in unbound mode.

The values for this property correspond to the the standard Visual Basic data type constants (vbInteger, vbLong, vbSingle, vbDouble, vbCurrency, vbDate, vbString, vbBoolean and vbByte)

## **DataType Property Applies To**

Column object



## DatabaseAction Property

Applies To

### Description

Determines the action taken when the Data Command button is pressed.

### Syntax

*object* . **DatabaseAction**[= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the action taken when the Data Command control is activated, as described in Settings.

### Settings

Setting	Description
0	Go to the first record in the table.
1	Go to the previous page of the table. The number of rows in a page is determined by the <b>PageValue</b> property.
2	Go to the previous record in the table.
3	Go to the next record in the table.
4	Go to the next page of the table. The number of rows in a page is determined by the <b>PageValue</b> property.
5	Go to the last record in the table.
6	Save a bookmark. Once saved, it is stored in the <b>SavedBookmark</b> property.
7	Goto a saved bookmark.
8	Refresh the record set.

There are constants available for the settings of this property.

### Remarks

In order for the **GotoBookmark** action to work, the bookmark must be set in the SavedBookmark property.

Create two SSDBCommand buttons; one captioned "Save Bookmark" with **DatabaseAction** = 6, and the other captioned "Goto Bookmark" with **DatabaseAction** = 7. In the **AfterClick** event for the Save Bookmark button, add the following code:

```
SSDBCommand2.SavedBookmark = SSDBCommand1.SavedBookmark
```

**Note:** This code assumes that the Save Bookmark button is SSDBCommand1 and the Goto Bookmark button is SSDBCommand2.

After scrolling through the database, you can click the "Goto Bookmark" button to return to the record whose bookmark you saved.

## **DatabaseAction Property Applies To**

SSDBCommand

## DefColWidth Property

Applies To

### Description

Sets or returns the default width used to initially display the column.

### Syntax

*object* . **DefColWidth**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the width used to initially display the column.

## **DefColWidth Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## DelayInitial Property

[See Also](#)

[Applies To](#)

### Description

Determines the amount of time (in milliseconds) before a repeatable button repeats the second click when the mouse button is held down.

### Syntax

*object* . **DelayInitial**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the amount of time (in milliseconds) to wait.

### Remarks

This is the amount of time between the mouse click and the moment repeating begins. Valid range for this property is 1 - 5000 with a default value of 500.

## **DelayInitial Property Applies To**

SSDBCommand

SSDBData

**DelayInitial Property See Also**

[DelaySubsequent](#)



## DelaySubsequent Property

[See Also](#)

[Applies To](#)

### Description

Determines the amount of time (in milliseconds) to wait until the third and subsequent clicks are repeated when the mouse button is held down on a repeatable button.

### Syntax

*object* . **DelaySubsequent**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the amount of time (in milliseconds) to wait.

### Remarks

Valid range for this property is 1 - 5000 with a default value of 100.

## **DelaySubsequent Property Applies To**

SSDBCommand

SSDBData

**DelaySubsequent Property See Also**

[DelayInitial](#)

## DeleteSelected Method

Applies To

### Description

Deletes all selected rows.

### Syntax

*object* . **DeleteSelected**

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.

### Remarks

This method will turn off the draw functions, delete the rows, and then turn the draw back on so that there is no flicker.

## **DeleteSelected Method Applies To**

SSDBGrid

[◀ Back](#)

## Displaying column totals from a DataGrid

There are two approaches to obtaining totals from data in a DataGrid. The simpler method is to step through each row in the grid and add the value of the cells in the column to be totaled. While easy to implement, this approach may run into serious performance problems on large data sets.

The second approach is to construct a total query that pulls the totals directly from the recordsource. The DataGrid is not involved in this procedure. This approach uses the underlying data engine, and may produce better performance. However, if the values in the data grid are changed, both the recordsource for the data and the recordsource for the totals must both be refreshed, and the controls updated, for the new data to be reflected in the totals.

Which would you like to do?

[Display totals by adding the values from the DataGrid](#)

[Display totals by performing a total query](#)

◀ Back

## Displaying one value and storing another

You can use a DataDropDown or a DataCombo to display the value of one field to the user, while storing the value from a different field "behind the scenes." The most common use of this technique is to present the user with a list of descriptive names to choose from, while storing only coded values for the list items in the database.

The following example uses the DBCombo to illustrate this process, but will work equally well with the DBDropDown

1. Place two Data controls on a form. Set the RecordSource of the first (Data1) to the table that contains the list of codes and their descriptions. Set the RecordSource of the second (Data2) to the table that will store the codes.
2. Place an SSDBCombo on the form. Change the **DataSource** property to 'Data2.'" This will link the edit portion of the control to the table that will store the coded values.
3. Set the **DataSourceList** property of the control to "Data1." This will link the list portion of the control to the table that supplies the codes and their descriptions.
4. Set the **DataField** property of the combo to the field in Data2 that will store the code.
5. Set the **DataFieldList** property of the combo to the field in Data1 that contains the coded values.
6. Set the **DataFieldToDisplay** property of the combo to the field in Data1 that contains the descriptions of the codes.

You may also want to format the drop-down portion of the combo using the Grid Editor.

When you drop down the combo and select a value from the displayed grid, the edit portion of the combo will display the value stored in the field indicated by **DataFieldToDisplay**. However, the value actually stored in the database will be the value taken from the field indicated by **DataFieldList**.

## Distributing Your Application

Once you have created a program using Data Widgets controls, you must distribute the OCX files with your application. There are no separate design time and runtime versions of the controls, therefore, the same OCX files you develop with can be shipped with your application.

<b>Filename</b>	<b>Description</b>
SSDATA16.OCX	16-Bit OCX containing SSDBData, SSDBOptSet, SSDBCommand
SSDATA32.OCX	32-Bit OCX containing SSDBData, SSDBOptSet, SSDBCommand
SSDATB16.OCX	16-Bit OCX containing SSDBGrid, SSDBDropDown, SSDBCombo
SSDATB32.OCX	32-Bit OCX containing SSDBGrid, SSDBDropDown, SSDBCombo

**Support files needed for distribution - 16 Bit**

**Support files needed for distribution - 32 Bit**



## DividerStyle Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the style of row divider used.

### Syntax

*object* . **DividerStyle**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the divider style, as described in Settings.

<b>Settings</b>	<b>Setting</b>	<b>Description</b>
0		Black line
1		(Default) Dark gray line
2		Raised
3		Inset
4		ForeColor

There are [constants](#) available for the settings of this property.

## **DividerStyle Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **DividerStyle Property See Also**

[DividerType](#)

## DividerType Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the type of row divider used.

### Syntax

*object* . **DividerType**[= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the divider type, as described in Settings.

Settings	Setting	Description
0	None	
1	Vertical	
2	Horizontal	
3	(Default) Both	

There are [constants](#) available for the settings of this property.

## **DividerType Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **DividerType Property See Also**

[DividerStyle](#)

## DoClick Method

Applies To

### Description

Fires the **Click** event.

### Syntax

*object* . **DoClick**

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.

### Remarks

The **DoClick** method simulates the user clicking the mouse.

## **DoClick Method Applies To**

SSDBCombo

SSDBCommand

SSDBDropDown

SSDBGrid



## DropDown Event

[See Also](#)

[Applies To](#)

### Description

Occurs when a dropdown drops down.

### Syntax

**Sub** control\_ **DropDown** ()

## **DropDown Event Applies To**

SSDBCombo

SSDBDropDown

## **DropDown Event See Also**

CloseUp event

## DropDownHwnd Property

[Applies To](#)

[Example](#)

### Description

Specifies the handle of the [Data DropDown](#) (Hwnd property) to be linked with the Data Grid.

### Syntax

*object* . **DropDownHwnd**[= *hwnd*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>hwnd</i>	A variant expression specifying the handle of the DataDropDown to be linked.

### Remarks

Set this property to the Hwnd property of a Data DropDown to enable a link between the grid and Data DropDown. The Data Grid will automatically display the dropdown button in a cell when it is active.

This property is only available at runtime.

The **DropDownHwnd** property can only be used with the Data DropDown control. It will **not** work with other controls.

## **DropDownHwnd Property Applies To**

Column object

## DroppedDown Property

[See Also](#)

[Applies To](#)

### Description

For [SSDBData](#), Determines whether the **ShowBookmarkDropdown** event will be fired.

For [SSDBCombo](#) and [SSDBDropDown](#), used in the **DropDown** event to cancel a dropdown from occurring.

For [SSDBCombo](#) and [SSDBGrid](#), causes the combo box to drop or close.

### Syntax

*object* . **DroppedDown**[= *boolean* ]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying the dropdown state, as described in Settings.

### Settings (SSDBData)

Setting	Description
<b>True</b>	<b>ShowBookmarkDropdown</b> event is fired.
<b>False</b>	<b>CloseBookmarkDropdown</b> event is fired.

### Settings (SSDBCombo/SSDBDropDown)

Setting	Description
<b>True</b>	When used in the <b>DropDown</b> event, cancels the dropdown.
<b>False</b>	No effect.

### Settings (SSDBCombo/SSDBGrid)

Setting	Description
<b>True</b>	Causes the combo box to drop.
<b>False</b>	Causes the combo box to close.

### Remarks

If you want to process the bookmark dropdown, set this value to **False** when you receive the ShowBookmarkDropdown event, otherwise, the control will display a bookmark dropdown list.

## **DroppedDown Property Applies To**

SSDBCombo

SSDBData

SSDBDropDown

SSDBGrid

## **DroppedDown Property See Also**

[CloseBookmarkDropdown](#) event

[ShowBookmarkDropdown](#) event





## Enhanced Data Control

[Properties](#)

[Events](#)

[Methods](#)

[FileNames](#)

[Collections](#)

[ObjectType](#)

[Objects](#)

The Enhanced Data Control (EDC) is an enhanced version of the Data Control that ships with Visual Basic. The EDC is used in conjunction with the data control rather than taking its place. The EDC can be oriented either horizontally or vertically, and can be sized to your liking at design time.

Some of the enhancements that the EDC provides include bookmark storage allowing you to return to a particular row at a later time, next page and previous page buttons, the ability to selectively enable/disable features of the EDC, and a speed button feature allowing the user to hold down a button to repeat its function.

[◀ Back](#)

[Adding the Enhanced Data Control](#)

[Anatomy of the Enhanced Data Control](#)

[Finding Information with the Enhanced Data Control](#)

[What are Bookmarks?](#)

[Speed Buttons](#)

[Binding to a Data Control Across Forms](#)

[◀ Back](#)

## **Enhanced Data Control Collections**

Bookmarks

[◀ Back](#)

## **Enhanced Data Control Events**

[AfterClick](#)

[Click](#)

[CloseBookmarkDropdown](#)

[CloseFindDialog](#)

[DbClick](#)

[DragDrop](#)

[DragOver](#)

[Error](#)

[FindResult](#)

[MouseDown](#)

[MouseMove](#)

[MouseUp](#)

[ShowBookmarkDropdown](#)

[ShowFindDialog](#)

[◀ Back](#)

## **Enhanced Data Control Methods**

[Drag](#)

[Find](#)

[WhereIs](#)

[◀ Back](#)

## **Enhanced Data Control Objects**

[Bookmark](#)

[◀ Back](#)

## **Enhanced Data Control Properties**

[\(About\)](#)

[\(Custom\)](#)

[Align](#)

[Alignment](#)

[BackColor](#)

[BalloonHelp](#)

[BevelColorFace](#)

[BevelColorFrame](#)

[BevelColorHighlight](#)

[BevelColorScheme](#)

[BevelColorShadow](#)

[BevelInner](#)

[BevelOuter](#)

[BevelWidth](#)

[BookmarkDisplay](#)

[BookmarksToKeep](#)

[BorderWidth](#)

[ButtonSize](#)

[Caption](#)

[CaptionAlignment](#)

[ColorMask](#)

[ColorMaskEnabled](#)

[DataField](#)

[DataSource](#)

[DelayInitial](#)

[DelaySubsequent](#)

[DragIcon](#)

[DragMode](#)

[DroppedDown](#)

[Enabled](#)

[FieldValue](#)

[FindBufferSize](#)

[FindDialog](#)

[FindFieldExclude](#)

[FindFieldInclude](#)

[Font](#)

[Font3D](#)

[ForeColor](#)

[Height](#)

[Index](#)



Left  
MouseIcon  
MousePointer  
Name  
Negotiate  
Orientation  
PageValue  
PictureButtons  
PictureCaption  
PictureCaptionAlignment  
PictureCaptionMetaHeight  
PictureCaptionMetaWidth  
RotateText  
RoundedCorners  
ShowAddButton  
ShowBookmarkButtons  
ShowCancelButton  
ShowDeleteButton  
ShowFindButtons  
ShowFirstLastButtons  
ShowPageButtons  
ShowPrevNextButtons  
ShowUpdateButton  
TabIndex  
Tag  
Top  
Visible  
WhatsThisHelpID  
Width  
WordWrap

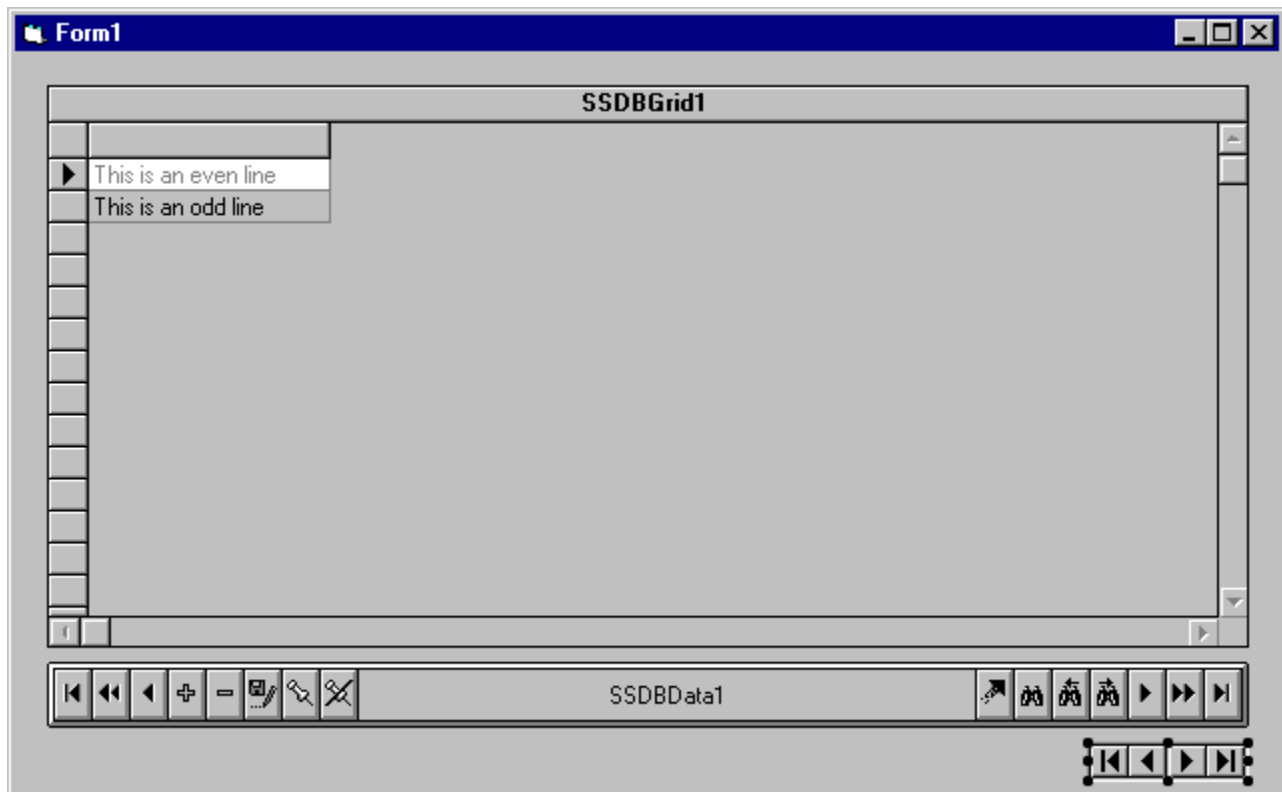
## Enhanced Data Control: Exercise 1

This section guides you through the creation of a sample program using the Enhanced Data control. For a complete description of this control, refer to the [Enhanced Data Control](#) section.

For these exercises, it is assumed that you have already launched your development application (Visual Basic), and that the control has been added to your toolbox. For more information on how to do this, refer to [Using Data Widgets](#).

You will create an application that makes use of the Enhanced Data control. The database used will be the BIBLIO.MDB that ships with Visual Basic and is located in the Visual Basic home directory. If you are using an environment other than Visual Basic, and do not have the BIBLIO.MDB database, consult [BIBLIO File Structure](#) for details on the file layout.

1. Place a standard data control on the form.
2. Set the **DatabaseName** property to 'BIBLIO.MDB' and set the **RecordSource** property to 'All Titles'. Set the **Visible** property to 'False'.  
This example will demonstrate how the Enhanced Data control supplements the standard data control by adding new navigational features. The standard Data Control is required for access to the database, but is not required for navigation.
3. Place a SSDBGrid control directly on the form by double clicking on the  tool in the Visual Basic toolbox.  
Resize the grid to a size that is suitable for your form.
4. Set the grid's **DataSource** property to 'Data1'
5. Place an Enhanced Data Control (EDC) directly on the form by double clicking on the  tool in the Visual Basic toolbox. Resize the EDC so that it fits the width of the form.

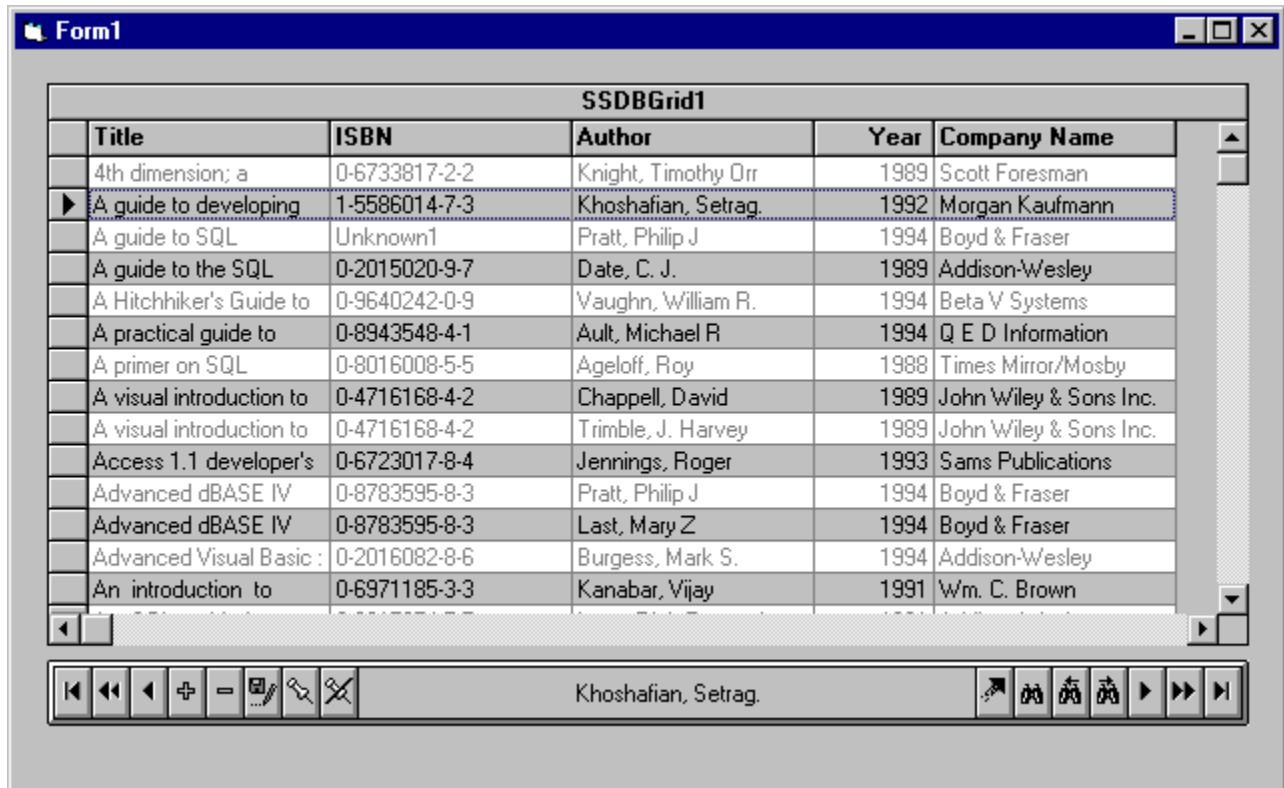


6. Set the EDC's **DataSource** property to 'Data1' and the **DataField** property to 'Author'.



This binds the EDC to the database. Remember that the EDC relies on the standard data control for access to the database. However, you can make the standard data control invisible so that your users do not see it at runtime.

7. Set the grid's **AllowAddNew** and **AllowDelete** properties to **True**. This allows you to add and delete records within the database by clicking the corresponding buttons on the EDC.
8. Run the application.



Try scrolling through the application. Use the following navigational buttons to help you:



**First Record**  
Jumps to the first record in the database.



**Previous Page**  
Jumps to the previous page in the database. A page is determined by the setting of **PageValue**.



**Previous Record**  
Jumps to the previous record in the database.



**Next Record**  
Jumps to the next record in the database.



**Next Page**  
Jumps to the next page in the database. A page is determined by the setting of **PageValue**.

**Last Record**

Jumps to the last record in the database.

Try adding and deleting records (you might want to make a backup copy of the database before you do this).

Use the following buttons to help you:

**Add Record**

Adds a new record to the end of the database.

**Delete Record**

Deletes the selected record from the database.


**Update Record**

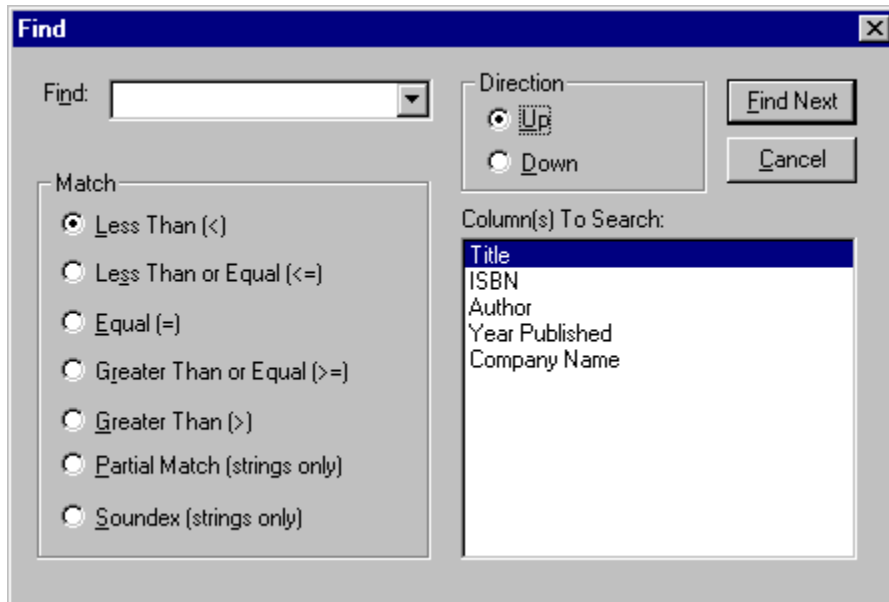
Updates the selected record in the database.



Don't delete this project, we'll need it in the next exercise.

## Enhanced Data Control: Exercise 2

One of the many useful features of the Enhanced Data Control is its searching capabilities. This exercise demonstrates some of the capabilities of the EDC's search routine. The project used in the last exercise should be running at this point.

1. To activate the Find dialog, click the  button. The Find dialog appears:



2. In the **Find** box, type 'John Wiley & Sons Inc.'.
3. Select 'Equal (=)' for the type of match.
4. Select 'Company Name' for the search column.
5. Click **Find Next**. The first occurrence of this text will be found.
6. To find the next occurrence of this text, click the **Find Next**  button.
7. To find the last occurrence of this text, click the **Find Previous**  button.





The **Find** dialog can be a very powerful tool for locating information in large databases. It also has advanced searching capabilities that include Soundex searching and Partial String searching. To try Soundex searching, replace the text in Step 2 with "Jon". To try Partial String searching, replace the text in Step 2 with "John"

**Note** The user can press the ESC key during an extensive search to exit.

## Enhanced Data Control: Exercise 3

Bookmarks are a powerful tool that allow you to "flag" a record in a database and later go back to it at the click of the mouse. This exercise demonstrates how you can use bookmarks to better manage your database. The project used in the last exercise should be running at this point.

Before we start, it is important to understand that using bookmarks is a two-step process. The first step is to Add the bookmark, the second step is to Goto the bookmark. You can only go to a bookmark that you have added.

1. Scroll through the database and select the first record you want to add a bookmark for.
2. Click the **Add Bookmark**  button. You will notice that two buttons (the Delete Bookmark and Goto Bookmark buttons) just became un-grayed and available for selection.
3. Scroll through the database and select another record you want to add a bookmark for.
4. Click the **Add Bookmark**  button.
5. Scroll through the database some more, and click the **Goto Bookmark**  button.
6. Select the bookmark you want to go to from the dropdown. Notice how the bookmark is represented by the field you are bound to. This is because bookmarks are binary and can not be displayed, so a text string must be associated with it. If you use the **Add** method to add bookmarks in code, you need to include the second parameter which is the display string.
7. Click the **Clear All Bookmarks**  button. All bookmarks that you created have just been deleted.

Now you can begin to see the power of the Enhanced Data Control. The three exercises we performed are a sampling of what you can do. The EDC also allows you to customize its button interface including the bitmap itself as well as selectively toggle buttons on or off.

## Enhanced Data Control Features

The Enhanced Data Control behaves as a front end to the Visual Basic data control adding new functionality such as bookmark navigation and page movement.

The following is a list of the features for this control:

- Re-position recordset by selecting from a dropdown list of up to 100 previously marked rows

- Buttons that perform database actions such as add, delete, update, plus bi-directional *n*th record paging

- Store and sort multiple bookmarks in the bookmark dropdown list

- Conditional and Soundex searching

- Optional user-defined pictures for each button

- Auto-repeat movement keys (forward, backward) plus bidirectional *n*th record paging

- 3D Font capability

- Caption text rotation

- Display/Hide any button

- Custom events give full programmatic control over button clicks and navigation.

## Error Event

[See Also](#)

[Applies To](#)

### Description

Occurs when there is an error updating the database with data from the control.

### Syntax

**Sub** control\_ **Error** (**ByVal** *DataError* **As Integer**, *Response* **As Integer**)

The event parameters are:

#### Parameter Description

---

*DataError*      The error number.

*Response*      A number corresponding to the response you want to take, as described in Settings.

The settings for *Response* are:

#### Value      Description

---

0                  Continue.

1                  (Default) Display the error message.

### Remarks

The **Error** event is fired whenever an error occurs while updating the database with data from the control.

You usually provide error-handling functionality for run-time errors in your code. However, run-time errors can occur when none of your code is running, as when:

- The Data control automatically opens a database and loads a Recordset object after the Form\_Load event.
- A control performs an operation such as the MoveNext method, the AddNew method, or the Delete method.

For more information on how to handle data-related errors, see "[How the Data Grid handles data validation and error checking.](#)"

## **Error Event Applies To**

SSDBGrid

## **Error Event See Also**

[AfterUpdate](#) event

[UpdateError](#) event

[How the Data Grid handles data validation and error checking](#)



## Error Messages

The following is a list of trappable errors that could occur at runtime when using the Data Widgets custom controls.

Error Number	Constant / <b>Error Text</b> / Description
30402	<i>ssItemNotInList</i> <b>"Item is not in list"</b> Validation determined that the current edit text is not in the associated dropdown list.
30403	<i>ssItemNotInList</i> <b>"Item is not in list"</b> Validation determined that the current edit text is not in the associated dropdown list.
30422	<i>ssItemNotInList</i> <b>"Item is not in list"</b> Validation determined that the current edit text is not in the associated dropdown list.
30423	<i>ssNullEdit</i> <b>"A null value is not allowed for this field"</b> Validation determined that the current edit text is null, but nulls are not allowed.
30424	<i>ssBadTypeEdit</i> <b>"The value is not of the correct type for this field"</b> Validation determined that the data type of the value currently in edit text does not match the data type of the associated list.
30425	<i>ssBitMap</i> <b>"PictureDropDown must be a bitmap"</b> You tried setting the PictureDropDown property to something other than a bitmap
30426	<i>ssDropItemsRange</i> <b>"Value must be from 1 to 100"</b> You tried setting the MaxDropDownItems or MinDropDownItems outside the allowed range.
30427	<i>ssMaxLessMinDropItem</i> <b>"MaxDropDownItems cannot be less than MinDropDownItems"</b> You tried setting MaxDropDownItems to a number smaller than the MinDropDownItems value.
30428	<i>ssdatbMinMoreMaxDropItem</i> <b>"MinDropDownItems cannot be greater than MaxDropDownItems"</b>

- 30430 You tried setting the MinDropDownItems to a value greater than the MaxDropDownItems.  
*ssdatbValue0to10*  
**"Value must be from 0 to 10"**  
You tried setting a property outside the allowed range.
- 30433 *ssdatbBadHost*  
**"Host environment does not support formatting"**  
You tried setting the BevelWidth property outside the allowed range.
- 30434 *ssdatbBadParam*  
**"Invalid parameter"**  
You tried passing an invalid parameter to a method.
- 30435 *ssdatbLevelCount*  
**"LevelCount must be from 1 to 10"**  
You tried setting a property outside the allowed range.
- 30436 *ssdatbFieldDelim*  
**"FieldDelimiter can be any one character or the word 'None'"**  
You tried setting FieldDelimiter to something other than that allowed.
- 30437 *ssdatbFieldSep*  
**"FieldSeparator can be any one character, the word 'Tab', or 'Space'"**  
You tried setting FieldSeperator to something other than that allowed.
- 30438 *ssdatbLevelNoGroup*  
**"The level count cannot be greater than zero if there are no groups"**  
You tried setting the property to something other than that allowed.
- 30439 *ssdatbReadOnlyRuntime*  
**"Property is read-only at runtime"**  
Property is read-only at run time. Its value cannot be set.
- 30440 *ssdatbInvalidWindowHandle*  
**"Invalid window handle"**  
You tried to set the DropDownhWnd property of a column in the DataGrid to something other than the window handle of a Data DropDown..
- 30441 *ssdatbWrongComboStyle*  
**"The combo style is of the wrong type (ie:**

**simple, dropdown, droplist)"**

You attempted to perform a combo box operation (add item, delete item, etc.) on a DataGrid column that was not set to a combo style.

30443

*ssdatbDataConversion*

**"Data type conversion error"**

A field type was specified for a Grid column in Unbound or AddItem mode, and data of an incorrect type was entered in that column.

30447

*ssdatbNameMustBeUnique*

**"Column name must be unique"**

The Name property of each column in a grid must be a unique value.

30448

*ssdatbCannotSetColsWithLay*

**"Cannot set Cols property when a layout is defined"**

Once you have used the Grid Editor to design a layout for the columns in a DataGrid, you cannot change the number of columns. Remove any layouts before attempting to change the number of columns in the grid.

32033

*ssEDCBevelInner*

**"BevelInner must be from 0 to 2"**

You tried setting the property to something outside the allowed range.

32034

*ssEDCBevelOuter*

**"BevelOuter must be from 0 to 2"**

You tried setting the property to something outside the allowed range.

32035

*ssEDCBevelWidth*

**"BevelWidth must be from 0 to 10"**

You tried setting the property to something outside the allowed range.

32036

*ssEDCBorderStyle*

**"BorderStyle must be either 0 or 1"**

You tried setting the property to something outside the allowed range.

32037

*ssEDCBorderWidth*

**"BorderWidth must be from 0 to 10"**

You tried setting the property to something outside the allowed range.

32038

*ssEDCRoundedCorners*

**"RoundedCorners must be either 0 or 1"**

You tried setting the property to something outside the allowed range.

32039

*ssEDCBevelColorScheme*

**"BevelColorScheme must be from 0 to 2"**

You tried setting the property to something outside the allowed range.

32040

*ssButtonSize*

**"ButtonSize must be from 5 to 100"**

You tried setting the property to something outside the allowed range.

32041

*ssPictureButtons*

**"PictureButtons must be of type Bitmap"**

You tried specifying a type other than Bitmap for PictureButtons.

32042

*ssShowBookmarkButtons*

**"ShowBookmarkButtons must be from 0 to 7"**

You tried setting the property to something outside the allowed range.

32043

*ssShowFirstLastButtons*

**"ShowFirstLastButtons must be either 0 or 1"**

You tried setting the property to something outside the allowed range.

32044

*ssShowPageButtons*

**"ShowPageButtons must be either 0 or 1"**

You tried setting the property to something outside the allowed range.

32045

*ssShowPrevNextButtons*

**"ShowPrevNextButtons must be either 0 or 1"**

You tried setting the property to something outside the allowed range.

32046

*ssShowAddButton*

**"ShowAddButton must be either 0 or 1"**

You tried setting the property to something outside the allowed range.

32047

*ssShowCancelButton*

**"ShowCancelButton must be either 0 or 1"**

You tried setting the property to something outside the allowed range.

32048

*ssShowDeleteButton*

**"ShowDeleteButton must be either 0 or 1"**

You tried setting the property to something outside the allowed range.

32049

*ssShowUpdateButton*

**"ShowUpdateButton must be either 0 or 1"**

- You tried setting the property to something outside the allowed range.
- 32050 *ssShowFindButtons*  
**"ShowFindButtons must be from 0 to 3"**
- You tried setting the property to something outside the allowed range.
- 32051 *ssEDCAalignment*  
**"Alignment must be from 0 to 8"**
- You tried setting the property to something outside the allowed range.
- 32052 *ssBookmarkDisplay*  
**"BookmarkDisplay must be from 0 to 2"**
- You tried setting the property to something outside the allowed range.
- 32053 *ssBookmarksToKeep*  
**"BookmarksToKeep must be from 1 to 100"**
- You tried setting the property to something outside the allowed range.
- 32054 *ssEDCCaptionAlignment*  
**"CaptionAlignment must be from 0 to 2"**
- You tried setting the property to something outside the allowed range.
- 32055 *ssColorMaskEnabled*  
**"ColorMastEnabled must be either 0 or 1"**
- You tried setting the property to something outside the allowed range.
- 32056 *ssPageValue*  
**"PageValue must be from 2 to 1000"**
- You tried setting the property to something outside the allowed range.
- 32057 *ssPictureCaptionAlignment*  
**"PictureCaptionAlignment must be from 0 to 14"**
- You tried setting the property to something outside the allowed range.
- 32058 *ssPictureCaptionMetaHeight*  
**"PictureCaptionMetaHeight must be a positive integer"**
- You tried setting the property to something outside the allowed range.
- 32059 *ssPictureCaptionMetaWidth*  
**"PictureCaptionMetaWidth must be a positive integer"**
- You tried setting the property to something outside the allowed range.

- 32060 *ssFindBufferSize*  
**"FindBufferSize must be from 1 to 1000"**  
You tried setting the property to something outside the allowed range.
- 32061 *ssOrientation*  
**"Orientation must be either 0 or 1"**  
You tried setting the property to something outside the allowed range.
- 32062 *ssEDCBalloonHelp*  
**"BalloonHelp must be either 0 or 1"**  
You tried setting the property to something outside the allowed range.
- 32065 *ssNoBookmarksLeft*  
**"There are no more bookmarks to remove"**  
You tried removing a bookmark when there are none.
- 32066 *ssCantMove*  
**"Illegal Move"**  
You tried moving to a location where there is no record.
- 32067 *ssNoCurrentRecord*  
**"No Current Record"**  
There is no current record in the database to which the control is bound. This may be caused by the database being at the beginning of the file before the first record, at the end of the file after the last record, or on a deleted record.
- 32068 *ssNoFindFieldsLeft*  
**"No fields remaining for find"**  
Your FindFieldExclude/FindFieldInclude properties have reduced the field count to zero. No fields are left to search.
- 32072 *ssDataChanged*  
**"Delete failed. Data in the underlying row has changed since the current row was fetched"**  
You attempted to delete a record using the Enhanced Data Control, but the record you are attempting to delete has changed in the underlying recordset.
- 32073 *ssUpdateInProgress*  
**"Delete failed. Update in progress"**  
You attempted to delete a record using the Enhanced Data Control while the record was being updated.
- 32074 *ssDeleteFailed*

**"Delete failed with error code <error code >"**

You attempted to delete a record using the Enhanced Data Control, and a non-specific data error occurred.

32080 *ssRemoveLastButton*

**"Cannot remove last button"**

You've attempted to remove the last button.

32081 *ssOptionNotUnique*

**"Option value must be unique"**

You've tried setting the option value to something that is already an option value for another button.

32082 *ssAlignment*

**"Alignment must be either 0 or 1"**

You tried setting the property to something outside the allowed range.

32083 *ssPictureAlignment*

**"PictureAlignment must be from 0 to 3"**

You tried setting the property to something outside the allowed range.

32084 *ssBevelColorScheme*

**"BevelColorScheme must be from 0 to 2"**

You tried setting the property to something outside the allowed range.

32085 *ssCols*

**"Cols must be from 1 to 10, but not greater than the total number of buttons."**

You tried setting the property to something outside the allowed range.

32086 *ssColWidth*

**"ColWidth must be a positive integer"**

You tried setting the property to something outside the allowed range.

32087 *ssFont3D*

**"Font3D must be from 0 to 4"**

You tried setting the property to something outside the allowed range.

32088 *ssHeightGap*

**"HeightGap must be from 1 to 1000"**

You tried setting the property to something outside the allowed range.

32089 *ssIndexSelected*

**"IndexSelected must be from 0 to the total number of buttons - 1"**

- You tried setting the property to something outside the allowed range.
- 32090 *ssMinColWidth*  
**"MinColWidth must be a positive integer equal to or greater than 30"**
- You tried setting the property to something outside the allowed range.
- 32091 *ssMinheight*  
**"MinHeight must be a positive integer equal to or greater than 15"**
- You tried setting the property to something outside the allowed range.
- 32092 *ssNumberOfButtons*  
**"NumberOfButtons must be from 0 to 99"**
- You tried setting the property to something outside the allowed range.
- 32093 *ssPictureMetaHeight*  
**"PictureMetaHeight must be a positive integer"**
- You tried setting the property to something outside the allowed range.
- 32094 *ssPictureMetaWidth*  
**"PictureMetaWidth must be a positive integer"**
- You tried setting the property to something outside the allowed range.
- 32095 *ssWidthGap*  
**"WidthGap must be a positive integer"**
- You tried setting the property to something outside the allowed range.
- 32096 *ssDCBBevelWidth*  
**"BevelWidth must be from 0 to 10"**
- You tried setting the property to something outside the allowed range.
- 32097 *ssDCBDelayValue*  
**"Delay Value must be from 1 to 5000"**
- You tried setting the property to something outside the allowed range.
- 32098 *ssDCBPageValue*  
**"PageValue must be from 2 to 1000"**
- You tried setting the property to something outside the allowed range.
- 32099 *ssDCBBorderStyle*  
**"BorderStyle must be either 0 or 1"**
- You tried setting the property to something



- outside the allowed range.
- 32100 *ssDCBCaptionAlignment*  
**"CaptionAlignment must be from 0 to 8"**  
You tried setting the property to something outside the allowed range.
- 32101 *ssDCBPictureAlignment*  
**"PictureAlignment must be from 0 to 14"**  
You tried setting the property to something outside the allowed range.
- 32102 *ssDCBDatabaseAction*  
**"DatabaseAction must be from 0 to 8"**  
You tried setting the property to something outside the allowed range.
- 32104 *ssDCBCantMove*  
**"Illegal Move"**  
You tried moving to a location where there is no record.

#### Even Row (Row 0)

[◀ Back](#)

#### Event Summary

	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

#### A

[AfterClick](#)  
[AfterColUpdate](#)  
[AfterDelete](#)  
[AfterInsert](#)  
[AfterPosChanged](#)  
[AfterUpdate](#)

#### B

[BeforeColUpdate](#)  
[BeforeDelete](#)  
[BeforeInsert](#)  
[BeforeRowColChange](#)  
[BeforeUpdate](#)  
[BtnClick](#)

#### C

[Change](#)  
[Click](#)

CloseBookmarkDropDown

CloseFindDialog

CloseUp

ColMove

ColResize

ColSwap

ComboCloseUp

ComboDropDown

## **D**

DropDown

## **E**

Error

## **F**

FindResult

## **G**

GrpHeadClick

GrpMove

GrpResize

GrpSwap

## **H**

HeadClick

## **I**

InitColumnProps

## **P**

PositionList

## **R**

RowColChange

RowLoaded

RowResize

## **S**

Scroll

ScrollAfter

SelChange

ShowBookmarkDropDown

ShowFindDialog

SplitterMove

## T

TextError

## U

UnboundAddData

UnboundDeleteRow

UnboundPositionData

UnboundReadData

UnboundWriteData

UpdateError

## V

ValidateList

## Extra Samples

The following samples illustrate how to implement commonly used features of Data Widgets 2.0. The samples are divided into groups by host environment.



### **Visual Basic 4.0**

Sample projects that illustrate more ways to use Data Widgets 2.0 in Visual Basic 4.0.



### **Visual C++ 4.0**

Information and samples showing how to use Data Widgets in Visual C++ using "wrapper" classes and the Microsoft Remote Data Control (MSRDC.)

## FieldDelimiter Property

[See Also](#)

[Applies To](#)

[Example](#)

### **Description**

Sets or returns the field delimiter used for an AddItem grid.

### **Syntax**

*object* . **FieldDelimiter**[=*string*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>String</i>	A string expression that evaluates to the field delimiter used in an AddItem grid.

### **Remarks**

The field delimiter is used to notify SSDBGrid of the start and end of a field. It is needed if your field contains the **FieldSeparator** character. At design time, a list of pre-defined delimiters is supplied, but you can enter your own. The default field delimiter is "none".

## **FieldDelimiter Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

**FieldDelimiter Property See Also**

[FieldSeparator](#)

## FieldLen Property

Applies To

### Description

Sets or returns the maximum column field length for editing.

### Syntax

*object* . **FieldLen**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the maximum column field length.

### Remarks

This property will determine the maximum amount of characters the user can type when editing a cell.

## **FieldLen Property Applies To**

Column object



## FieldSeparator Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the field separator used for an AddItem grid.

### Syntax

*object* . **FieldSeparator**[= *text*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>text</i>	A string expression that evaluates to the field separator used in an AddItem grid.

### Remarks

The field separator is used to notify [SSDBGrid](#) of the separation of two fields. At design time, a list of pre-defined separator is supplied, but you can enter your own. The default field separator is "tab".

## **FieldSeparator Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

**FieldSeparator Property See Also**

[FieldDelimiter](#)

## FieldValue Property

Applies To

Example

### Description

Returns the field value for the active record.

### Syntax

*object* . **FieldValue**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the field value.

### Remarks

FieldValue contains the value of the field specified in **DataField** for the active record.

## **FieldValue Property Applies To**

SSDBData

## Find Method

[See Also](#)      [Applies To](#)      [Example](#)

### Description

Finds a specified string in the database.

### Syntax

*object*. **Find**(*FindString* **as Variant**, *Criteria* **As Variant**, *Direction* **As Variant**, *ColToSearch* **As Variant**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>FindString</i>	Specifies the string to search for..
<i>Criteria</i>	Determines the criteria to use for the search, as described in Settings.
<i>Direction</i>	Determines the direction to search the database, as described in Settings.
<i>ColToSearch</i>	The index of the database column to search.

### Settings

The settings for *Criteria* are:

<b>Setting</b>	<b>Description</b>
1	Less Than
2	Less Than or Equal To
3	Equal To
4	Greater Than
5	Greater Than or Equal To
6	Partial Match

There are [constants](#) available for the settings of this parameter.

The settings for *Direction* are:

<b>Setting</b>	<b>Description</b>
1	Down (Next)
2	Up (Previous)

There are [constants](#) available for the settings of this parameter.

## **Find Method Applies To**

SSDBData

## **Find Method See Also**

[CloseFindDialog](#) event



## FindBufferSize Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the number of records read into the buffer for find operations.

### Syntax

*object* . **FindBufferSize**[= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the number of records to be read into the buffer for find operations.

### Remarks

When you initiate a search for information, the [Enhanced Data Control](#) reads *number* records into memory, and searches for the information. If it does not find the information, it reads in another *number* records, repeating this process until it either finds the data or reaches the end.

Generally speaking, when searching a large database, it is best to keep *number* set to a number that decreases the amount of disk accesses, such as 1000.

## **FindBufferSize Property Applies To**

SSDBData

## **FindBufferSize Property See Also**

[FindDialog](#)

[ShowFindButtons](#)

## FindDialog Property

[See Also](#)

[Applies To](#)

### Description

Determines whether the Find dialog is displayed.

### Syntax

*object* . **FindDialog**[= *boolean* ]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the Find dialog is displayed, as described in Settings.

Settings	Setting	Description
	<b>True</b>	The Find dialog is displayed.
	<b>False</b>	Has no effect.

### Remarks

Provides the ability to fire the **ShowFindDialog** and **CloseFindDialog** events. By setting this property to **True**, you will, in effect, activate **ShowFindDialog**. When processing is complete, you should set this to **False** to activate **CloseFindDialog**. This is only available at runtime.

## **FindDialog Property Applies To**

SSDBData

## **FindDialog Property See Also**

[FindBufferSize](#)

[ShowFindButtons](#)

## FindFieldExclude Property

[See Also](#)

[Applies To](#)

### Description

Determines which fields will not be displayed in the Find dialog box.

### Syntax

*object* . **FindFieldExclude** = *fieldname* [;*fieldname* ][;*fieldname* ] [...]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>fieldname</i>	A string expression that evaluates to the name of a field in the recordset that is bound to the control.

### Remarks

This property gives you the ability to specify which fields will not be available for the user to select in the Find dialog box. By default, all searchable fields are available. By specifying the name of one or more fields for this property, you can prevent interactive searches from using those fields.

This property will accept the name of a single field or a list of field names delimited by semicolons or commas. Field names will be parsed by the control, and any field that is unavailable or spelled incorrectly will generate an error. If the property is passed multiple field names and only some of them are incorrect, the correct ones will still be unavailable in the Find dialog.

This property works in conjunction with the **FindFieldInclude** property to determine which fields will be available for searching. The control will take the following steps to determine which fields to include in the dialog:

1. If the **FindFieldInclude** property is blank, then all fields are included in the list of possible search fields. If **FindFieldInclude** is not blank, then only those fields in the **FindFieldInclude** list are included in the list of possible search fields.
2. If the **FindFieldExclude** property is blank, then no further field processing occurs. If **FindFieldExclude** is not blank, then all the fields in the string are excluded from the list of possible search fields. Therefore, fields passed to the **FindFieldExclude** property will be unavailable for searching, even if they are specified in the **FindFieldInclude** property.

## **FindFieldExclude Property Applies To**

SSDBData



## **FindFieldInclude Property See Also**

[FindFieldInclude](#)

## FindFieldInclude Property

[See Also](#)

[Applies To](#)

### Description

Determines which fields will be displayed in the Find dialog box.

### Syntax

```
object . FindFieldInclude = fieldname [;fieldname ][:fieldname ] [...]
```

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>fieldname</i>	A string expression that evaluates to the name of a field in the recordset that is bound to the control.

### Remarks

This property gives you the ability to specify which fields will be available for the user to select in the Find dialog box. By default, all searchable fields are available. By specifying the name of one or more fields for this property, you can restrict interactive searches to just those fields.

This property will accept the name of a single field or a list of field names delimited by semicolons or commas. Field names will be parsed by the control, and any field that is unavailable or spelled incorrectly will generate an error. If the property is passed multiple field names and only some of them are incorrect, the correct ones will still appear in the dialog.

This property works in conjunction with the **FindFieldExclude** property to determine which fields will be available for searching. The control will take the following steps to determine which fields to include in the dialog:

1. If the **FindFieldInclude** property is blank, then all fields are included in the list of possible search fields. If **FindFieldInclude** is not blank, then only those fields in the **FindFieldInclude** list are included in the list of possible search fields.
2. If the **FindFieldExclude** property is blank, then no further field processing occurs. If **FindFieldExclude** is not blank, then all the fields in the string are excluded from the list of possible search fields. Therefore, fields passed to the **FindFieldExclude** property will be unavailable for searching, even if they are specified in the **FindFieldInclude** property.

## **FindFieldInclude Property Applies To**

SSDBData

## **FindFieldInclude Property See Also**

[FindFieldExclude](#)

## FindResult Event

Applies To

### Description

Occurs after a search has been completed.

### Syntax

**Sub** control\_FindResult(*vBookmark* **As Variant**, *RtnDispErrMsg* **As Integer**)

The event parameters are:

### Parameter Description

---

*vBookmark* A variant expression containing the bookmark associated with the row which contains the text found.

*RtnDispErrMsg* An integer expression that indicates if an error message box should be displayed.

### Remarks


If there was no match, *vBookmark* will be empty. The EDC will then display a message box indicating that no match was found, unless it is overridden by setting *RtnDispErrMsg* to **False**.

## **FindResult Event Applies To**

SSDBData

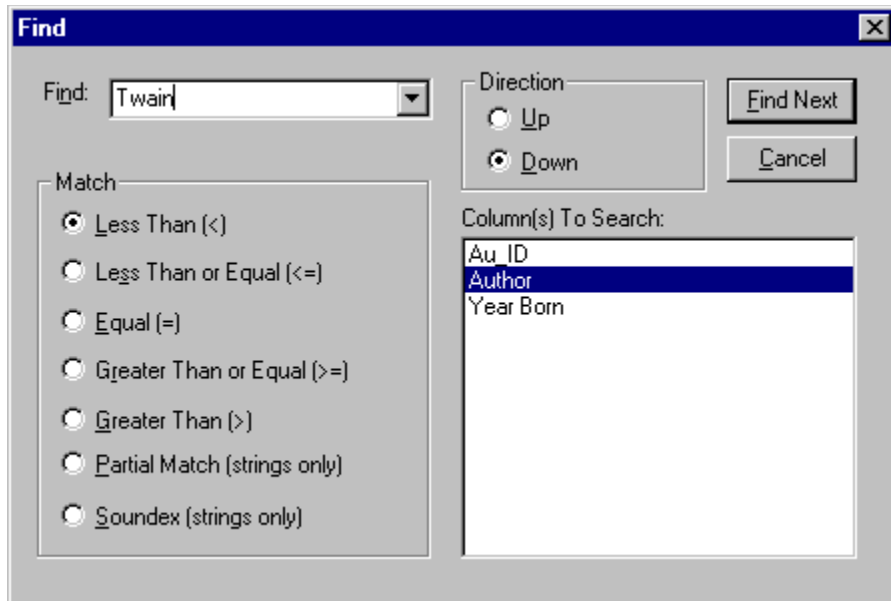
## Finding Information with the Enhanced Data Control


Finding information in a database field is a snap with the Enhanced Data Control.

To activate the Find dialog, click the  button. You can use the

 and

 buttons to continue searching once you've found a match.



Find	Specify the data to search for. A list of recent searches is available by clicking the  button.
Direction	Specifies the direction in which to search. Selecting <b>Down</b> will search from the current point to the end of the database. Selecting <b>Up</b> will search from the current point to the start of the database.
Less Than	Match only if the text entered in the <i>Find</i> dialog is less than the value in the database. Examples of this are 1 < 2 and APPLE < BEAR
Less Than or Equal To	Match only if the text entered in the <i>Find</i> dialog is less than or equal the value in the database. Examples of this are 2 <= 2 and APPLE <= BEAR
Equal To	Match only if the text entered in the <i>Find</i> dialog equals the value in the database. Examples of this are 5 = 5 and DOG = DOG
Greater Than or Equal	Match only if the text entered in the <i>Find</i> dialog equals or exceeds the value in the database. Examples of this are 7 >= 2 and DOT >= DOS
Greater Than	Match only if the text entered in the <i>Find</i> dialog exceeds the value in the database. Examples of this are 10 > 9 and TREE > BARK.
Partial	Match only if a portion of the string specified in the

- Match *Find* dialog matches a portion in the database. An example of this is specifying "Eng" in the *Find* dialog and returning "Engine" and "England". This works for strings only.
- Soundex Match only if the string sounds like one in the database. An example of this is specifying "Skool" and returning "School". This works for strings only.

Note: The user can press the ESC key during an extensive search to exit.



## FirstRow Property

Applies To

### Description

Sets or returns the bookmark for the first visible row.

### Syntax

*object* . **FirstRow**[= *variant*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>variant</i>	A variant expression containing the bookmark of the first visible row.

### Remarks

This property is not available at design time. Setting this property makes the specified row the first visible row in the grid.

## **FirstRow Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## Font Object

[See Also](#)

[Applies To](#)

[Example](#)

### Description

The Font object contains information needed to format text on the control.

### Properties

---

[Bold](#)

[Size](#)

[Italic](#)

[Strikethrough](#)

[Name \(Default\)](#)

[Underline](#)

### Remarks

You frequently identify a **Font** object using the **Font** property of an object that displays text.

## Font Object Applies To

SSDBCombo

SSDBCommand

SSDBData

SSDBDropDown

SSDBGrid

SSDBOptSet

## Font Object See Also

Bold

Font3D

Italic

Name

Size

Strikethrough

Underline

## Font3D Property

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Determines the 3D style of the caption text for the control.

### Syntax

*object* . **Font3D**[= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the 3D font style, as described in Settings.

Settings	Description
0	(Default) None. Caption is displayed flat (not three-dimensional). This is the default setting.
1	Raised with light shading. Caption appears as if it is raised slightly off the screen.
2	Raised with heavy shading. Caption appears even more raised.
3	Inset with light shading. Caption appears as if it is inset slightly into the screen.
4	Inset with heavy shading. Caption appears even more inset.

There are [constants](#) available for the settings of this property.

### Remarks

The **Font3D** works in conjunction with the **Font** property. Settings 2 and 4 (heavy shading) look best with larger, bolder fonts. Dramatic effects can be created by combining the different **Font3D** settings with other **Font** properties.

## **Font3D Property Applies To**

SSDBCombo

SSDBCommand

SSDBData

SSDBDropDown

SSDBGrid

SSDBOptSet

## **Font3D Property See Also**

[Caption](#)



## Fonts

Fonts are supported through the **Font** object. At design time fonts are set through one of the font properties (For example: **Font**). Depending on the development environment you are using, a dialog box containing font information may be available so that you can set properties of the **Font** object. If not, you can set the font properties through the Property Pages. The following properties are supported by the **Font** object:

### Properties

---

<u>Bold</u>	<u>Size</u>	<u>Underline</u>
<u>Italic</u>	<u>StrikeThrough</u>	
<u>Name</u>		

Fonts can be set either through the font dialog at design time or by setting properties of the **Font** object at runtime.

## ForeColorEven Property

[See Also](#)

[Applies To](#)

### Description

Determines the row's foreground (text) color for even-numbered rows.

### Syntax

*object* . **ForeColorEven**[= *color*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>color</i>	A long integer value or constant that determines the color.

## **ForeColorEven Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **ForeColorEven Property See Also**

[ForeColor](#)

[ForeColorOdd](#)

## ForeColorOdd Property

[See Also](#)

[Applies To](#)

### Description

Determines the row's foreground (text) color for odd-numbered rows.

### Syntax

*object* . **ForeColorOdd**[= *color*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>color</i>	A long integer value or constant that determines the color.

## **ForeColorOdd Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **ForeColorOdd Property See Also**

[ForeColor](#)

[ForeColorEven](#)

## GetBookmark Method

Applies To

Example

### Description

Returns a bookmark of a row relative to the current row.

### Syntax

*object* . **GetBookmark**( [*Row* **As Long**])

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Row</i>	A long value specifying the row offset of the bookmark to get.



## **GetBookmark Method Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## Grid Editor

The grid editor is used to easily customize the appearance of the Data Grid, Data Combo, and Data DropDown controls. Through a tabbed dialog, you can define the number of columns and groups as well as their appearance and associated properties.

The Grid Editor uses a work area called the Design Grid that simulates how your Data Grid will appear. The Design Grid works in much the same way as the Data Grid with the ability to move, resize, and swap columns and groups.

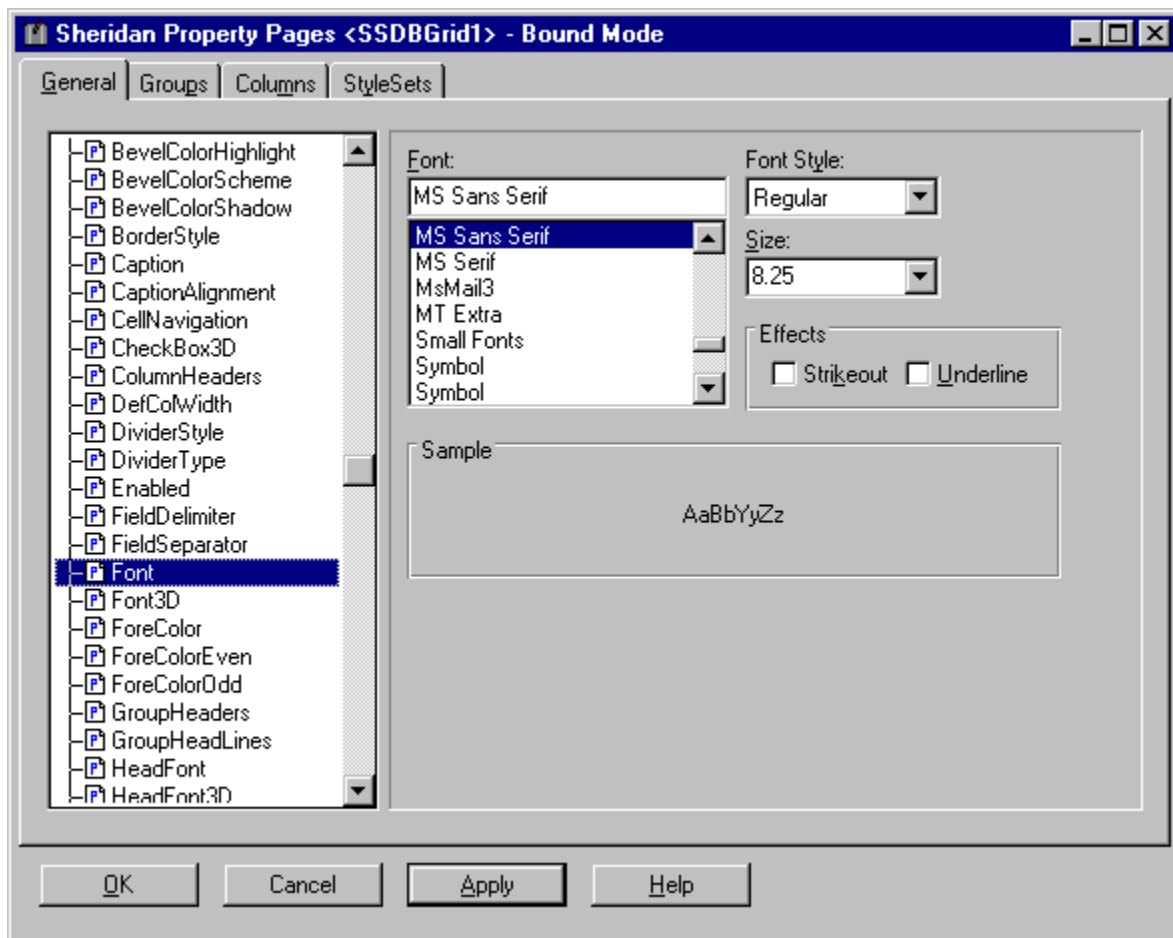
### Accessing the Grid Editor

#### General Tab

#### Groups Tab

#### Columns Tab

#### StyleSets tab



## Grid Editor: Accessing

The Grid Editor is activated by selecting the **(Custom)** property from the properties list, or by selecting "Properties" from the right-click menu of Visual Basic. You can use the Grid Editor at any time in design time to make changes to your grid. The Grid Editor simulates layout by displaying a grid known as the Design Grid.

As a demonstration, the Grid Editor (GRIDEDIT.EXE) can be executed as a standalone program. In this case, you will be able to select **Open** from the *File* menu and select a database to use. All functions of the grid can be explored.

Clicking the **OK** button applies the changes you have made and exits the Grid Editor.

Clicking the **Apply** button applies the changes you have made and remains in the Grid Editor.

Clicking **Cancel** aborts the changes you have made and exits the Grid Editor.

## Grid Editor: Columns tab

The Columns Tab allows you to define the columns that appear in your grid. Columns appear in the Design Grid, allowing you to visualize how your grid will look at runtime.

## Resizing

The width of the grid or the selected column can be changed by entering a value (in twips) in the text boxes labeled "Grid Width" and "Column Width".

Alternatively, you can resize the width of the grid by dragging the splitter, and you can resize the width of a column by clicking on the right-edge of its header and dragging the column to the desired size.

## Adding a column to the Design Grid

1. Click the **Add** button.
2. Specify the name for the column in the "Add Column" dialog. The column will be added to the grid.

## Removing a column from the Design Grid

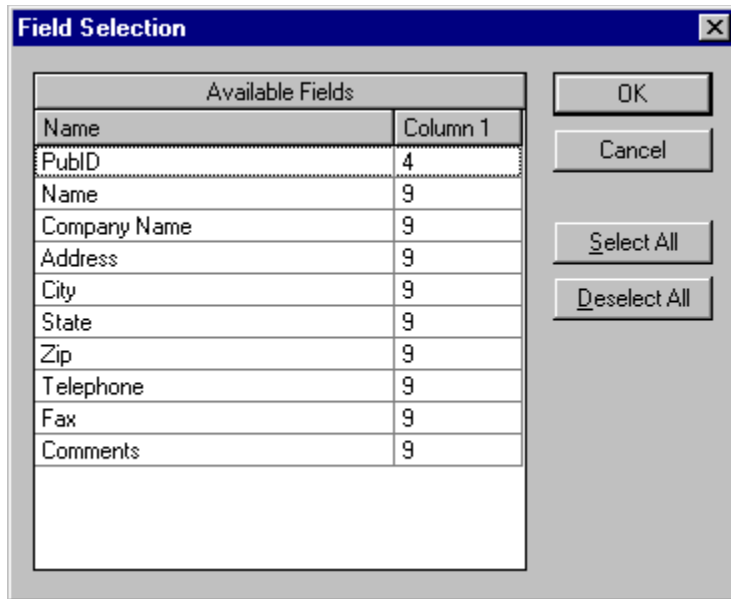
1. In the Design Grid, click the header of the column to remove.
2. Click the **Remove** button.

**Note:** It is possible to remove multiple columns at once. Simply click on each header corresponding to the column you want removed.

## Adding columns to the Design Grid from a bound datasource

It is possible to automatically create columns based on the field structure of a database that the grid is bound to. It is possible to automatically create columns based on the field structure of a database that the grid is bound to. To add columns from a bound datasource:

1. Click the **Fields** button. The "Field Selection" dialog appears listing all fields in the database.



2. Select the fields you want to appear as columns.  
To select all fields in the database, click the **Select All** button.
3. Click the **OK** button.  
The selected fields appear as columns in the Design Grid.

## Grid Editor: General tab

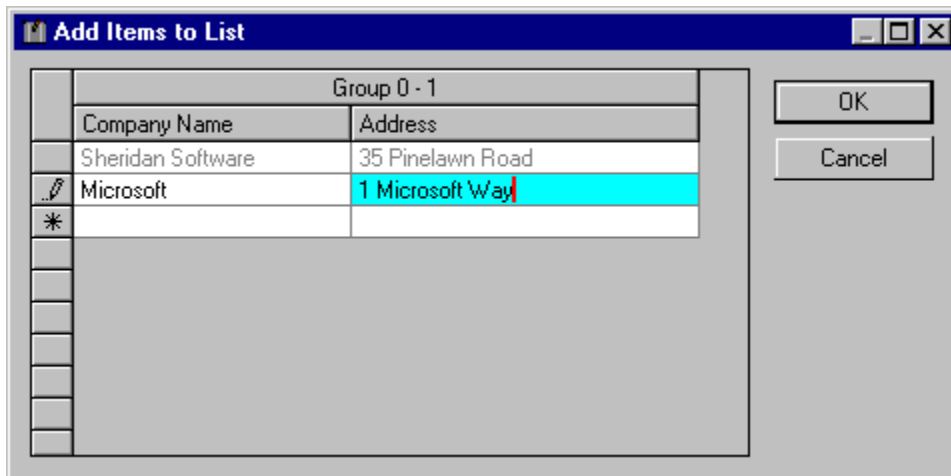
The General tab has the appearance of a standard Sheridan Property Page. Through a tree structure, you are able to select and modify properties that apply to the grid as a whole. To modify a property, simply select it from the tree and make the desired changes from the options presented on the right.

There are two items on the General Tab that need special explanation; (Add Items...) and StyleSets.

Adding items to an AddItem grid

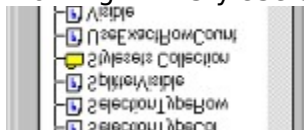
If **DataMode** is set to AddItem, the (Add Items...) option appears on the top of the tree. Selecting this option allows you to manually fill an AddItem grid with data.

Click the **Add Items** button to add data. The "Add Items To List" dialog appears:



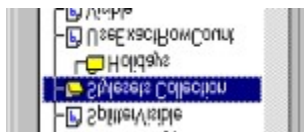
Fill in the data as needed, clicking the **OK** button when you are finished. Clicking the **Cancel** button allows you to exit without saving your changes.

Working with StyleSets

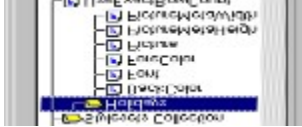


The Grid Editor allows you to easily maintain and apply StyleSets. If you are not familiar with StyleSets, you should first read about [StyleSets](#). Before you can apply a StyleSet, you must first define it. You can have an unlimited amount of StyleSets for any given grid, however, StyleSets are not interchangeable between grids.

When you first access the Grid Editor's *General* Tab, the StyleSets Collection will be collapsed by default. To expand the collection, double-click it. This will display any StyleSets that have already been created:



To see the individual properties applicable to the StyleSet, double-click that StyleSet:



To modify a property within a StyleSet, simply select it from the tree and make the desired changes from the options presented on the right.

## Adding StyleSets

1. Select StyleSets from the tree structure.
2. Click the **Add** button that appears to the right.
3. Specify a name in the "Add StyleSet" dialog.
4. The StyleSet now appears in the tree:



## Removing (Deleting) StyleSets

1. Select the StyleSet to remove from the tree structure.
2. Click the **Remove** button that appears to the right.  
The selected StyleSet is deleted.

Applying StyleSets takes place in the [StyleSets tab](#).

## Grid Editor: Groups tab

The Groups Tab allows you to define the groups that appear in your grid. Groups allow you to logically arrange fields that are associated with one another. For example, you could have a group called "Address Information" that contains the Address, City, State, and Zip Code fields from a database. Groups appear in the Design Grid, allowing you to visualize how your grid will look at runtime.

## Resizing

The width of the grid or the selected group can be changed by entering a value (in twips) in the text boxes labeled "Grid Width" and "Group Width".

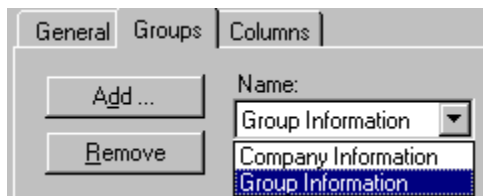
Alternatively, you can resize the width of the grid by dragging the splitter, and you can resize the width of a group by clicking on the right-edge of its header and dragging the group to the desired size.

## Adding a group to the Design Grid

1. Click the **Add** button.
2. Specify the name for the group in the "Add Group" dialog. The group will be added to the grid.

## Removing a group from the Design Grid

1. Select the group from the *Name* drop-down list.



2. Click the **Remove** button.

## Working with group properties

There are certain properties that are group-specific. These properties can be easily changed through the Grid Editor.

To set group specific properties:

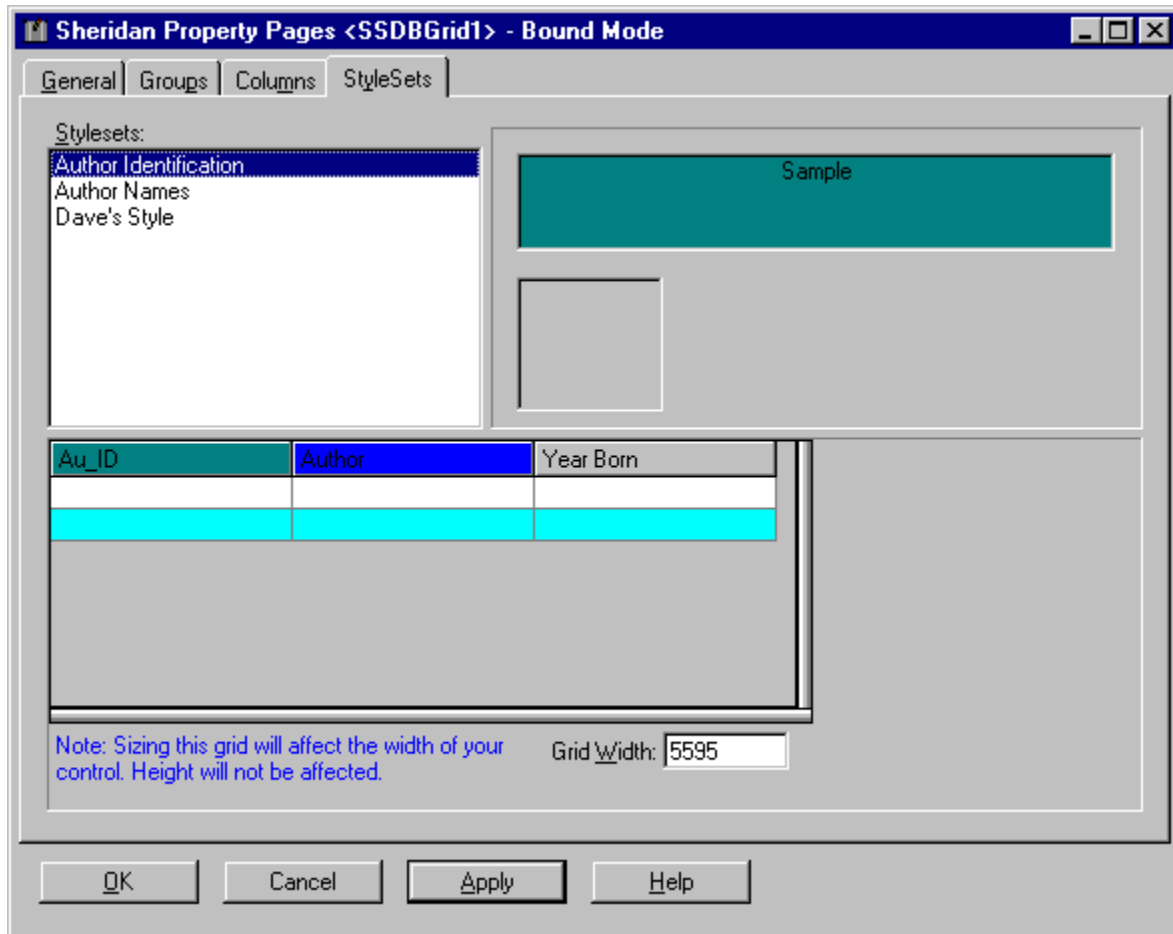
1. Select the group from the *Name* drop-down list.
2. Select the property to modify from the tree and make the desired changes from the options presented on the right.



## Grid Editor: StyleSets tab

With the StyleSets Tab, you are able to apply the StyleSets you have created. Select the StyleSet you want to use from the list, and drag it to the part of the Design Grid you want it applied to. For more information on StyleSets, refer to the [StyleSet](#) Property.

When you select a StyleSet, a sample of the attributes it has will appear to the right. Similarly, when you apply a StyleSet, you will see it in the Design Grid.



## Grid Selector

## **Group Header**

## Group Object

Applies To

### Description

The group object represents a group within a grid.

### Properties

---

<u>AllowSizing</u>	<u>HeadForeColor</u>	<u>Top</u>
<u>Caption</u>	<u>HeadStyleSet</u>	<u>Visible</u>
<u>CaptionAlignment</u>	<u>Left</u>	<u>Width</u>
<u>ColPosition</u>	<u>Position</u>	
<u>HasHeadBackColor</u>	<u>Selected</u>	
<u>HasHeadForeColor</u>	<u>StyleSet</u>	
<u>HeadBackColor</u>	<u>TagVariant</u>	

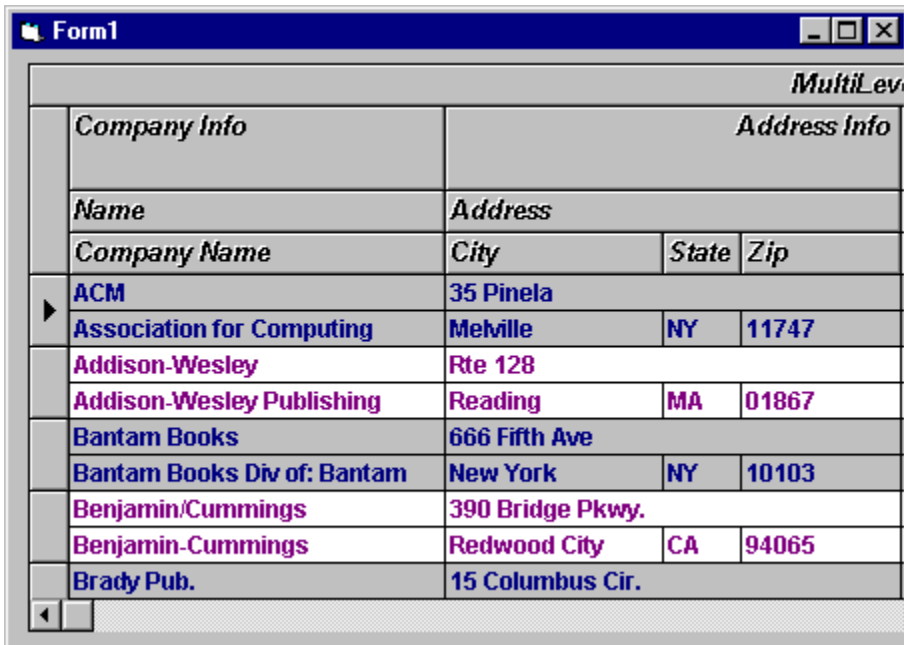
### Objects

---

Column

### Remarks

The **Column** object is available via the **Group** object as well as directly. The value of the index changes when accessing **Groups** through the **Column** object.



<i>MultiLevel</i>			
<i>Company Info</i>	<i>Address Info</i>		
<i>Name</i>	<i>Address</i>		
<i>Company Name</i>	<i>City</i>	<i>State</i>	<i>Zip</i>
ACM	35 Pinela		
Association for Computing	Melville	NY	11747
Addison-Wesley	Rte 128		
Addison-Wesley Publishing	Reading	MA	01867
Bantam Books	666 Fifth Ave		
Bantam Books Div of: Bantam	New York	NY	10103
Benjamin/Cummings	390 Bridge Pkwy.		
Benjamin-Cummings	Redwood City	CA	94065
Brady Pub.	15 Columbus Cir.		

In the example grid above, the following is true when accessing the **Column** object directly:

```
? SSDBGrid1.Columns(5).Caption  
Zip
```

Accessing the **Column** object through the **Group** object yields:

```
? SSDBGrid1.Groups(1).Columns(3).Caption  
Zip
```

In the top example, the sixth column is requested whereas in the bottom example, the third column in the second group is requested.

In the top example, you are requesting the sixth item of the entire grid. In the bottom example, you are requesting the fourth item from the second group.

## **Group Object Applies To**

Groups collection

## Group Property

[See Also](#)

[Applies To](#)

### Description

Returns the current group.

### Syntax

*object* . **Group**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the current group.

## **Group Property Applies To**

SSDBGrid



## **Group Property See Also**

Col

Grp

Row

## GroupHeadLines Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the number of lines to display for group headers.

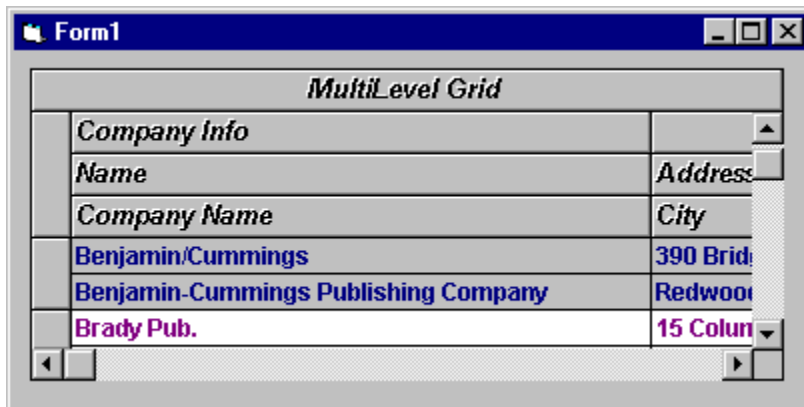
### Syntax

*object* . **GroupHeadLines**[= *number*]

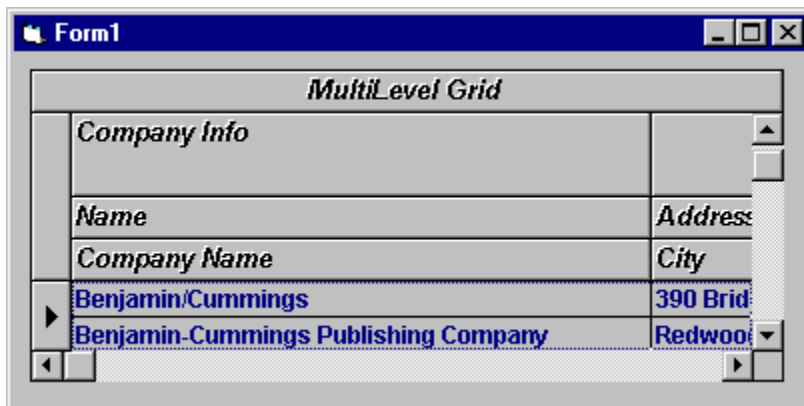
Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the number of lines to display for group headers.

### Remarks

The following examples demonstrate the use of the GroupHeadLines property:



This is an example of GroupHeadLines = 1. The group header "Company Info" is displayed in 1 row.



This is an example of GroupHeadLines = 2. The group header "Company Info" is displayed in 2 rows.

## **GroupHeadLines Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

**GroupHeadLines Property See Also**

[HeadLines](#)

## GroupHeaders Property

[See Also](#)

[Applies To](#)

### Description

Determines whether group headers will be displayed.

### Syntax

*object* . **GroupHeaders**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether group headers will be displayed, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) Group headers will be displayed.
<b>False</b>	Group headers will not be displayed

## **GroupHeaders Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **GroupHeaders Property See Also**

[ColumnHeaders](#)

## Groups Collection

[See Also](#)

[Applies To](#)

[Example](#)

### Description

The groups collection represents a set of data grid group objects.

### Properties

---

[Count](#)

[Item](#)

### Methods

---

[Add](#)

[Remove](#)

[RemoveAll](#)



## **Groups Collection Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **Groups Collection See Also**

Group object

## Groups Method

Applies To

### Description

Returns group object at specified index.

### Syntax

*object* . **Groups**( [*Index* **As Variant**])

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Index</i>	A variant specifying the group number.

### Remarks

When no index is specified the group object is returned.

## **Groups Method Applies To**

SSDBCombo

SSBBDropDown

SSDBGrid

## Grp Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the current group.

### Syntax

*object* . **Grp**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the group that the column belongs to.

### Remarks

The **Grp** property is useful for returning the group number that the column is a member of. It can also be used to set the group number that the column belongs to.

## **Grp Property Applies To**

Column object

## **Grp Property See Also**

Col

Group

Row

## GrpContaining Method

Applies To

### Description

Returns the group number located at the specified x-coordinate.

### Syntax

*object* . **GrpContaining**( *X* **As Single**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>X</i>	A floating point value specifying the X-coordinate



## **GrpContaining Method Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## GrpHeadClick Event

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Occurs when a group heading is clicked on.

### Syntax

```
Sub control_GrpHeadClick ([GrpIndex As Integer])
```

**The event parameters are:**

#### **Parameter Description**

---

*GrpIndex*      The group number being clicked on.

## **GrpHeadClick Event Applies To**

SSDBGrid

## **GrpHeadClick Event See Also**

[GrpResize](#) event

[HeadClick](#) event

## GrpMove Event

[See Also](#)

[Applies To](#)

### Description

Occurs before a group is moved.

### Syntax

**Sub** control\_**GrpMove** (*GrpIndex* **As Integer**, *NewPos* **As Integer**, *Cancel* **As Integer**)

### The event parameters are:

<b>Parameter</b>	<b>Description</b>
<i>GrpIndex</i>	The group number being moved.
<i>NewPos</i>	An integer expression that specifies the visual position the group is being moved to.
<i>Cancel</i>	An integer expression that specifies whether the operation occurs.

### Remarks

The **GrpMove** event is fired after a user moves a group, but before the move is redrawn. You can cancel this event from occurring by setting *Cancel* to true.

Swapping or moving groups does not change the name or number of the group, that is group number 2 is still group number 2 despite being moved.

## **GrpMove Event Applies To**

SSDBGrid

## **GrpMove Event See Also**

AfterPosChanged event

ColMove event

ColSwap event

GrpSwap event

## GrpPosition Method

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Returns the index of the group relative to the collection.

### Syntax

*object* . **GrpPosition**(*grpPos* **As Integer**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>grpPos</i>	An integer expression specifying the position of the group as it appears visually.

### Remarks

Remember that groups can be moved, swapped, or made invisible, so the order they appear in is not always their order in the collection.



## **GrpPosition Method Applies To**

SSDBGrid

## **GrpPosition Method See Also**

ColPosition method

Position property

## GrpResize Event

[See Also](#)

[Applies To](#)

### Description

Occurs before a group is resized.

### Syntax

**Sub** control\_**GrpResize** (*GrpIndex* **As Integer**, *Cancel* **As Integer**)

The event parameters are:

#### Parameter Description

---

*GrpIndex*      The group number being resized.

*Cancel*        An integer expression that specifies whether the operation occurs.

### Remarks

The **GrpResize** event is fired after a user resizes a group, but before the resize is redrawn.

## **GrpResize Event Applies To**

SSDBGrid

## **GrpResize Event See Also**

[GrpHeadClick](#)

[HeadClick](#)

[ResizeWidth](#)

## GrpSwap Event

[See Also](#)

[Applies To](#)

### Description

Occurs before a group is swapped.

### Syntax

```
Sub control_GrpSwap (GrpIndex As Integer, NewPos As Integer, Cancel As Integer)
```

### The event parameters are:

<b>Parameter</b>	<b>Description</b>
<i>GrpIndex</i>	The group number being moved.
<i>NewPos</i>	An integer expression that specifies the visual position the group is being swapped to.
<i>Cancel</i>	An integer expression that specifies whether the operation occurs.

### Remarks

The **GrpSwap** event is fired after a user swaps a group, but before the swap is redrawn. You can cancel this event from occurring by setting *Cancel* to true.

Swapping or moving groups does not change the name or number of the group, that is group number 2 is still group number 2 despite being moved.

## **GrpSwap Event Applies To**

SSDBGrid

## **GrpSwap Event See Also**

AfterPosChanged event

ColMove event

ColSwap event

GrpMove event





## Guided Tours



### **Data Grid** control

Sample programs using the Data Grid (Chapter 5)

Exercise 1: [Creating a Bound Data Grid](#)

Exercise 2: [Creating an Unbound Data Grid](#)

Exercise 3: [Creating an AddItem data Grid](#)



### **Data Combo** Control

Sample programs using the Data Combo (Chapter 6)

Exercise 1: [Creating an application using the Data Combo](#)

Exercise 2: [Customizing the Data Combo](#)



### **Data DropDown** Control

Sample program using the Data DropDown (Chapter 7)

Exercise 1: [Creating an application using the Data DropDown](#)



### **DataOptionSet** Control

Sample program using the Data DropDown (Chapter 8)

Exercise 1: [Creating an application using the DataOptionSet](#)



### **Enhanced Data Control** Control

Sample programs using the Enhanced Data Control (Chapter 9)

Exercise 1: [Creating an application using the EDC](#)

Exercise 2: [Using the Find feature of the EDC](#)

Exercise 3: [Using Bookmarks in the Enhanced Data Control](#)



### **Data Command** Control

Sample program using the Data Command Button (Chapter 10)

Exercise 1: [Creating an application using the Data Command](#)

[Biblio File Structure](#)

## HasBackColor Property

[See Also](#)

[Applies To](#)

### Description

Returns or sets whether the column has a background color.

### Syntax

*object* . **HasBackColor**[ = *boolean* ]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the column has its own background color, as described in Settings.

### Settings

Setting	Description
<b>True</b>	Column has its own background color.
<b>False</b>	Column does not have its own background color.

### Remarks

The background color can be set on a control level (`control.BackColor = color`) that applies to all columns in the grid or on an object level (`control.object(number).BackColor = color) that affects only the specified column. When the column object has its own, HasBackColor will be set to true.`

Setting **HasBackColor** to *false* causes the column's backcolor to revert back to that defined for the control.

The **HasBackColor** property is needed because setting **BackColor** to 0 will cause the color to be black, and not disabled as is the case with most other properties.

## **HasBackColor Property Applies To**

Column object

**HasBackColor Property See Also**

[HasForeColor](#)

## HasForeColor Property

[See Also](#)

[Applies To](#)

### Description

Returns or sets whether the column has its own foreground color.

### Syntax

*object* . **HasForeColor**[= *boolean* ]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the column has its own foreground color, as described in Settings.

### Settings

Setting	Description
<b>True</b>	Column has its own foreground color
<b>False</b>	Column does not have its own foreground color.

### Remarks

The foreground color can be set on a control level (`control.ForeColor = color`) that applies to all columns in the grid or on an object level (`control.object(number).ForeColor = color) that affects only the specified column. When the column object has its own, HasForeColor will be set to true.`

Setting **HasForeColor** to *false* causes the column's forecolor to revert back to that defined for the control.

The **HasForeColor** property is needed because setting **ForeColor** to 0 will cause the color to be black, and not disabled as is the case with most other properties.

## **HasForeColor Property Applies To**

Column object

**HasForeColor Property See Also**

[HasBackColor](#)

## HasHeadBackColor Property

[See Also](#)

[Applies To](#)

### Description

Returns or sets whether the header has a background color specified.

### Syntax

*object* . **HasHeadBackColor**[= *boolean* ]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the header has its own background color, as described in Settings.

### Settings

Setting	Description
<b>True</b>	Header has a background color specified.
<b>False</b>	Header does not have a background color specified.

### Remarks

Setting **HasHeadBackColor** to *false* causes the object's **HeadBackColor** to revert back to the default setting.



## **HasHeadBackColor Property Applies To**

Column object

Group object

## **HasHeadBackColor Property See Also**

[HasHeadForeColor](#)

[HeadBackColor](#)

[HeadForeColor](#)

## HasHeadForeColor Property

[See Also](#)

[Applies To](#)

### Description

Returns or sets whether the header has a foreground color specified.

### Syntax

*object* . **HasHeadForeColor**[= *boolean* ]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the header has a foreground color specified, as described in Settings.

### Settings

Setting	Description
<b>True</b>	Header has a foreground color specified.
<b>False</b>	Header does not have a foreground color specified.

### Remarks

Setting **HasHeadForeColor** to *false* causes the object's **HeadForeColor** to revert back to the default setting.

## **HasHeadForeColor Property Applies To**

Column object

Group object

## **HasHeadForeColor Property See Also**

[HasHeadForeColor](#)

[HeadBackColor](#)

## HeadBackColor Property

[See Also](#)

[Applies To](#)

### Description

Determines the header's background color.

### Syntax

*object* . **HeadBackColor**[= *color* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>color</i>	A long integer value or constant that determines the background color of the header.

## **HeadBackColor Property Applies To**

Column object

Group object

## **HeadBackColor Property See Also**

[HasHeadBackColor](#)

[HasHeadForeColor](#)

[HeadForeColor](#)



## HeadClick Event

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Occurs when a column heading is clicked on.

### Syntax

```
Sub control_HeadClick ([ColIndex As Integer])
```

**The event parameters are:**

<b>Parameter</b>	<b>Description</b>
------------------	--------------------

---

<i>ColIndex</i>	The column number being clicked on.
-----------------	-------------------------------------

## **HeadClick Event Applies To**

SSDBGrid

## **HeadClick Event See Also**

GrpResize event

HeadClick event

## HeadFont Object

[See Also](#)

[Applies To](#)

[Example](#)

### Description

The HeadFont object contains information needed to format header and caption text on a grid.

### Properties

---

[Bold](#)

[Size](#)

[Italic](#)

[Strikethrough](#)

[Name \(Default\)](#)

[Underline](#)

### Remarks

You frequently identify a **HeadFont** object using the **HeadFont** property of an object that displays text. At design time, **HeadFont** is shown in the properties list and acts as a property, allowing you to select the font name to be used.

## **HeadFont Object Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **HeadFont Object See Also**

[Bold](#)

[Font3D](#)

[Italic](#)

[Name](#)

[Size](#)

[Strikethrough](#)

[Underline](#)

## HeadFont3D Property

Applies To

### Description

Determines the 3D style of the caption and header text for the control.

### Syntax

*object* . **HeadFont3D**[= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the 3D font style, as described in Settings.

### Settings

Setting	Description
0	(Default) None. Text is displayed flat (not three-dimensional).
1	Raised with light shading. Caption appears as if it is raised slightly off the screen.
2	Raised with heavy shading. Caption appears even more raised.
3	Inset with light shading. Caption appears as if it is inset slightly into the screen.
4	Inset with heavy shading. Caption appears even more inset.

There are constants available for the settings of this property.

### Remarks

The **HeadFont3D** property works in conjunction with the **HeadFont** property. Settings 2 and 4 (heavy shading) look best with larger, bolder fonts. Dramatic effects can be created by combining the different **HeadFont3D** settings with other **HeadFont** properties.

## **HeadFont3D Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid



## HeadForeColor Property

[See Also](#)

[Applies To](#)

### Description

Determines the column header foreground (text) color.

### Syntax

*object* . **HeadForeColor**[= *color*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>color</i>	A long integer value or constant that determines the foreground color of the column header.

## **HeadForeColor Property Applies To**

Column object

Group object

## **HeadForeColor Property See Also**

[HasHeadBackColor](#)

[HasHeadForeColor](#)

[HeadBackColor](#)

## HeadLines Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the number of lines to display for column headers.

### Syntax

*object* . **HeadLines**[= *number*]

Part	Description
------	-------------

<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
---------------	---

<i>number</i>	An integer expression specifying the number of lines to display for column headers.
---------------	---

### Remarks

This example demonstrates the use of the **HeadLines** property:

<i>Company Info</i>	
<i>Name</i>	<i>Address</i>
<i>Company Name</i>	<i>City</i>
ACM	Association New York
Bantam Books	666 Fifth Av
Bantam Books Div of: Bantam Doubleday Dell	New York
Benjamin/Cummings	390 Bridge
Benjamin-Cummings Publishing Company	Redwood C
Brady Pub.	15 Columel
Bantam Books Div of Bantam Doubleday Dell	New York

This is an example of `HeadLines = 1`. The column headers (Name, Company Name, Address, City) each span one row.

<i>Company Info</i>	
<i>Name</i>	<i>Address</i>
<i>Company Name</i>	<i>City</i>
<b>ACM</b>	<b>Associati</b>
	<b>New York</b>
<b>Bantam Books</b>	<b>666 Fifth Av</b>
<b>Bantam Books Div of: Bantam Doubleday Dell</b>	<b>New York</b>
<b>Benjamin/Cummings</b>	<b>390 Bridge</b>

This is an example of HeadLines = 2. The column headers (Name, Company Name, Address, City) each span two rows.

## **HeadLines Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **HeadLines Property See Also**

[GroupHeadLines](#)

## HeadStyleSet Property

[See Also](#)

[Applies To](#)

### Description

Returns or sets the name of a **HeadStyleSet** in the **StyleSets** collection.

### Syntax

*object* . **HeadStyleSet**[= *text*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>text</i>	A string expression that evaluates to the name of a HeadStyleSet.

### Remarks

This property determines the HeadStyleSet to be used for the controls and objects listed in the *Applies To* list. Note that the **HeadStyleSet** specified must be in the **StyleSets** collection and its properties must be set before it can be used. If a change is made to a HeadStyleSet, the control must be refreshed.

HeadStyleSets will override each other based on the following hierarchy:

#### Group Area:

**Group**.HeadStyleSet (overrides all below)  
**Control**.HeadStyleSet

#### Column Area:

**Column**.HeadStyleSet (overrides all below)  
**Group**.HeadStyleSet (overrides all below)  
**Control**.HeadStyleSet

The following is a list of properties used in the various HeadStyleSets.

### Properties Used by SSDBCombo, SSDBDropDown, SSDBGrid

***For Group Heading Area, Record Selectors, Column Heading Area, and Caption Area:***

---

BackColor          Font          ForeColor

***For Caption Area:***

---

Picture                  PictureMetaHeight    PictureMetaWidth

### Properties Used by the Group Object:



***For Group Heading Area and Column Heading Area:***

---

BackColor                  Font                  ForeColor

***For Group Heading Area:***

---

Picture                  PictureMetaHeight      PictureMetaWidth

**Properties Used by the Column Object:**

***For Column Heading Area:***

---

ForeColor                  Font                  PictureMetaHeight  
BackColor                  Picture                  PictureMetaWidth

## **HeadStyleSet Property Applies To**

Column object

Group object

SSDBCombo

SSDBDropDown

SSDBGrid

## **HeadStyleSet Property See Also**

[HeadStyleSet](#)

[StyleSet](#) object

[StyleSets](#) collection

## HeightGap Property

Applies To

### Description

Determines the amount of vertical distance between option buttons.

### Syntax

*object* . **HeightGap**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the amount of vertical distance between option buttons.

### Remarks

Valid range is 0 to 32767 with a default value of 2. Setting the value to 0 causes the option buttons to have no vertical gap between each other.

## **HeightGap Property Applies To**

SSDBOptSet

## How To Apply Pictures To Cells

This procedure demonstrates how to apply pictures to all the cells in a column, based on a user or program initiated event. You could also apply the pictures by default by using only the code in the `Form_Load` and the `RowLoaded` events, and using only the `True` condition from the `IF` statement in `RowLoaded`.

1. First, you must create a flag that will be used to determine whether to apply the picture to the cells:

```
(General) (declarations)
Dim bFlag As Boolean
```

2. Then create a `StyleSet` that includes the picture you wish to apply. You should also set the flag to its initial state:

```
Private Sub Form_Load()
    SSDBGrid1.StyleSets.Add "Picture"
    SSDBGrid1.StyleSets("Picture").Picture = LoadPicture("ARROW.BMP")
    bFlag = False
End Sub
```

3. In the **RowLoaded** event, apply the `StyleSet` to the individual cell, or clear the cell's `StyleSet`, based on the value of the flag:

```
Private Sub SSDBGrid1_RowLoaded(ByVal Bookmark As Variant)
    If bFlag = True Then
        SSDBGrid1.Columns(1).CellStyleSet "Picture"
    Else
        SSDBGrid1.Columns(1).CellStyleSet ""
    End If
End Sub
```

4. Finally, set the flag to the desired state, then perform a **Refresh** on the grid to initiate the change.

```
Private Sub Command1_Click()
    bFlag = Not bFlag
    SSDBGrid1.Refresh
End Sub
```

[◀ Back](#)

## Data Widgets How-To Help

### Which task would you like to accomplish?

Attach a DataGrid to a memory array

Add pictures to cells

Perorm calculations (such as totals) on the values in a DataGrid

Include an unbound column in a bound DataGrid

Have a drop-down store a different value than the one displayed

Clear all formatting and selection information from a grid

## How the Data Grid Handles Data Validation and Error Checking

See Also

Whenever data has been modified in a row of the data grid and the user takes an action to update the row, error checking is performed. This error checking validates the data to ensure it is valid for insertion into the database based on database rules and field type. You can also take advantage of this feature to implement your own data validation and error handling code. If a data error occurs and you do not provide code to handle it, the error message will be passed to the end user, which may not be desirable.

When a record of grid data is updated, the grid first checks each column one at a time to see if the data is valid according to the rules and field types of the underlying database. If data for any field is invalid, the **UpdateError** event will be fired. You can use this event to try and correct the data entry problem, based on the error value returned.

How you manage an error that occurs in **UpdateError** determines the sequence of events that will follow. If you modify the value of the column that caused the error, no error condition will exist and the control will continue to check the remaining columns. If you do not modify the value of the offending column, an error condition will be created, and the control will not verify any remaining columns.

**Note** If you cancel the **UpdateError** event, *no further processing will take place*. None of the events described beyond this point will occur.

Note that at this point, the data is still held in the control's buffer; nothing has been written to the database. What happens next depends on whether an error condition exists. If there is no error condition, the control will attempt to update the database. If there has been a column-level error that was not corrected, the control will not attempt to update the database.

Next the control will fire the **AfterUpdate** event. **AfterUpdate** will be fired if:

- The database was successfully updated
- The database update failed
- The **UpdateError** event completed with errors and no update was attempted.

**AfterUpdate** receives a True or False parameter (*RtnDispErrorMsg*) to indicate whether the update of the database succeeded. A value of True indicates the update failed.

This is the first place you may implement specific error-handling code to prevent the end user from seeing an error message. (While the **UpdateError** event returns error codes for the programmer, it will not display an end-user error message or halt processing without programmer intervention.) Any database-specific errors, such as trying to update a locked record, can be trapped in the **AfterUpdate** event. In general, you should implement code to handle column-level errors in the **UpdateError** event, and deal with database-level errors in the **AfterUpdate** event.

If any kind of an error occurs while updating the database, the **Error** event is also fired. The **Error** event is similar to the one found in the grid control that ships with Visual Basic 4, and returns the same type of error code. The **Error** event is the final place for you to trap any update errors and prevent the error message from being passed to the end user. Note that the **Error** event is general and is also fired by other controls under different circumstances.

If the error is not trapped in either the **AfterUpdate** or **Error** events, an error message is displayed to the end user.



## How the DataGrid Handles Null Values

See Also

If you delete all the text from a cell, the cell contains no value. Different database systems handle this situation in different ways. Some databases allow **null values** (nulls) to be stored in the database. Other DBMSes cannot accept null values and will return an error or refuse to update the record if a null value is attempted for a field.

To deal with these situations, the DataGrid takes special action when you update a row that contains an empty cell. The grid will first query the back-end database to see if it can accept a null value. If it can, the null will be stored in the database. If the back-end database cannot accept null values, the DataGrid will store an empty string ("") in the database.

Note that this applies only to String type fields.

## **HwndEdit Property Applies To**

SSDBCombo

SSDBGrid

## **Important Note on Using VC++ Examples**

If you are using Data Widgets 2.0B or have Internet Explorer 4.0, there are some changes that need to be made to your existing Visual C++ code. The method names in the wrapper classes now have an \_ (underscore) character inserted at the beginning. You will need to change the uses of the method accordingly. One approach is to simply remove the underscore from the beginning of the method name in the wrapper class automatically generated for you, and then refer to the method by the original name.

Also, MFC puts up an unnecessary assertion in its `COleControl::InitiazlizeIID`'s function if the original classid is not found in the typelib. The problem does not affect release versions and you can ignore the assertion in debug builds. You can get rid of the assertion by editing the .mak and .rc files changing the old GUID entries to the new ones.

## Included Files

The Setup program will place OCX files in the directories you have specified. Sample applications from the manual are located in the \SAMPLES subdirectory of the Data Widgets home directory.

The following table gives a brief description of the files that may have been installed on your hard disk during the Setup process. Data Widgets selectively installs support files based on the version numbers of files already installed on your system.

<b>Filename(s)</b>	<b>Description</b>
COMDLG16.OCX	16-Bit Common Dialog OCX (used for demo programs)
COMPOBJ.DLL	Support DLL
DAO2516.DLL	Support DLL
DATW1TO2.HLP	Version 1.0 to 2.0 Conversion help file.
DATWFAQ.HLP	Frequently Asked Questions help file.
MFC40.DLL	Support DLL (Microsoft Foundation Class DLL)
MFC42.DLL	Support DLL (Microsoft Foundation Class DLL)
MFCO40.DLL	Support DLL (Microsoft Foundation Class DLL)
MFCANS32.DLL	Support DLL (Microsoft Foundation Class DLL)
MSAJT200.DLL	Support DLL (compatibility layer DLL)
MSJETERR.DLL	Support DLL (compatibility layer DLL)
MSJETINT.DLL	Support DLL (compatibility layer DLL)
MSOUTL16.OCX	16-Bit Outline Control OCX (used for demo programs)
MSVC40.DLL	Support DLL (Microsoft VC DLL)
MSVCRT.DLL	Support DLL (Microsoft VC DLL)
MSVCRT40.DLL	Support DLL (Microsoft VC DLL)
OC30.DLL	Support DLL (32-Bit data binding DLL)
OLE2.DLL	Support DLL (OLE DLL)
OLE2DISP.DLL	Support DLL (OLE DLL)
OLE2NLS.DLL	Support DLL (OLE DLL)
OLEPRO32.DLL	Support DLL (OLE DLL)
OLE2PROX.DLL	Support DLL (OLE DLL)
OLE2CONV.DLL	Support DLL (OLE DLL)
OLE2.REG	Support file (OLE registration file)
README.WRI	Data Widgets late-breaking information
SCP.DLL	Support DLL
SELECT.BMP	SSDBGrid row selector bitmap
SSCMD16.EXE	16-Bit SSDBCommand custom property pages
SSCMD32.EXE	32-Bit SSDBCommand custom property pages
SSDATA16.OCX	16-Bit OCX containing SSDBData, SSDBOptSet, SSDBCommand
SSDATA32.OCX	32-Bit OCX containing SSDBData, SSDBOptSet,

	SSDBCommand
SSDATB16.OCX	16-Bit OCX containing SSDBGrid, SSDBDropDown, SSDBCombo
SSDATB32.OCX	32-Bit OCX containing SSDBGrid, SSDBDropDown, SSDBCombo
SSDATWD2.HLP	Data Widgets Online Help
SSDBHPIC.BMP	SSDBData button bitmap
SSDBHPC2.BMP	SSDBData button bitmap
SSDODEMO.EXE	DataOptionSet demo
SSDOS16.EXE	16-Bit SSDBOptSet custom property pages
SSDOS32.EXE	32-Bit SSDBOptSet custom property pages
SSEDC16.EXE	16-Bit SSDBData custom property pages
SSEDC32.EXE	32-Bit SSDBData custom property pages
SSGRID16.EXE	16-Bit SSDBGrid Layout Editor
SSGRID32.EXE	32-Bit SSDBGrid Layout Editor
SSPP16.DLL	16-Bit Property Pages DLL
SSPP32.DLL	32-Bit Property Pages DLL
STDOLE.TLB	Support File
STORAGE.DLL	Support DLL (System DLL)
TABCTL16.OCX	16-Bit Tab Control OCX (used for demo programs)
THREED16.OCX	16-Bit 3D Control OCX (used for demo programs)
TYPELIB.DLL	Support DLL (type library DLL)
UNBOUND.MDB	Unbound grid sample database file
UNINSTALL.EXE	Data Widgets 2.0 uninstall program
VAEN2.DLL	Support DLL
VAEN21.OLB	Support DLL
VBAJET.DLL	Support DLL (compatibility layer DLL)
VBEN16.DLL	Support DLL
VB40016.DLL	16-Bit Visual Basic Runtime DLL
VB40032.DLL	32-Bit Visual Basic Runtime DLL
VBDB16.DLL	Support DLL (Visual Basic database DLL)
\SAMPLES	Directory containing projects demonstrated in Chapters 5 - 10

[◀ Back](#)

### Including an unbound column in a bound DataGrid

You can easily add unbound columns to a bound Data Grid (or other grid-like control) using the Grid Editor of the property pages, or through code.

I want to add the column at design time using the Grid Editor  
[Show me how to add an unbound column in code](#)

## IndexSelected Property

Applies To

### Description

Sets or returns the selected option button. Valid range is 0 to  $n$  where  $n$  is the index of the last button. Once a button is selected, button-specific property settings will affect only that button.

### Syntax

*object* . **IndexSelected**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the button number in a SSDBOptSet control.

### Remarks

This property can be used at Design time to set properties for any of the buttons in a DataOptionSet.

Index numbering of option buttons begins at 0. If there is no button selected, or if no buttons exist, the value will be -1.

**IndexSelected** allows you to easily determine the selected button.

## **IndexSelected Property Applies To**

SSDBOptSet

## InitColumnProps Event

Applies To

Example

### Description

Occurs when the grid is initially loaded, allowing the setting of group and column properties.

### Syntax

```
Sub control_InitColumnProps ()
```



## **InitColumnProps Event Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## Inserting controls into your VC++ project

To insert the control into your project initially use the Component Gallery. Click on the OLE Control tab, and you will see a list of the OLE controls available on your system. Select the icon for the control you want to include. A dialog should appear offering to add the necessary wrapper classes. Click OK to generate these classes.

Depending on the version of VC++ you are using, the Generate Wrapper Classes dialog may not appear. You then need to generate the additional wrapper classes manually. To do this, open Class Wizard and go to the Class Info tab. Click the Add Class button and select "From An OLE TypeLib". Change the filter in the Open File dialog to All files (\*.\*) then locate the OCX control that you are using, select it, and click OK. For example, to use the SSDBGrid, you would choose SSDATB32.OCX. When you select this file, ClassWizard will retrieve the necessary wrapper classes for use in your project.

In working with OLE controls you need to work extensively with Variants. Microsoft has a class called COleVariant, which makes working with variants easier. You can either use a COleVariant or work with the VARIANT structure.

```
//If you are not going to use COleVariant you need to initialize
//the variant type to VT_EMPTY
//VARIANT va;
//va.vt = VT_EMPTY;

COleVariant va;
LPDISPATCH lpdisp;

//Use empty variant to get dispatch to Columns Collection
lpdisp = m_grid.Columns(va);

//Use ISSColumns constructor, pass it the dispatch pointer
ISSColumns cols(lpdisp);

//Initialize a Variant using m_grid.GetCol() to return
//the current column.  VARIANT vacol;
vacol.vt = VT_I2;
vacol.iVal = m_grid.GetCol();
//Use the columns collection you created to get a dispatch
//pointer to a single column
lpdisp = cols.Item(vacol);
//use that dispatch pointer to create the single column
ISSColumn col(lpdisp);
//now you can access the member functions of ISSColumn
AfxMessageBox(col.GetText());
```

Select a related topic:

[Object Concepts](#)

[Property Pages](#)

[Fonts](#)

## Introduction to OCX controls

[Related Topics](#)

### What is an OCX control?

An OCX control is a specific type of program that makes use of *Object Linking and Embedding* (OLE) to provide functions to other programs. Because it gives programs something they did not originally have, an OCX control is known as an OLE *server*, and the program that uses its services is an OLE *client*. OCX controls can provide a nearly unlimited range of functions to their clients.

### How is an OCX control different from a VBX control?

The VBX control specification was designed exclusively for use with Visual Basic. Although some other languages offer limited VBX support, the majority of VBX controls function only in Visual Basic. VBX controls are also limited in other ways. Their 16-bit architecture restricts their ability to use memory and to function in a 32-bit operating system, such as Windows 95 or Windows NT.

The difference between OCX and VBX controls may not even be apparent to you if you program exclusively in Visual Basic. You access the properties of an OCX control at design time and through code just as you do the properties of a VBX. The process of including both types of controls in your project and distributing them is very similar. The similarities end when you move outside of the Visual Basic programming environment.

OCX controls are supported by a much wider range of platforms, including other languages, database management systems, and productivity applications. OCX controls can be used as the building blocks in a modular software environment, where a complete project might include your own code, custom controls and commercial applications all working together. OCX controls also have the ability to make full use of the newest 32-bit operating systems, taking advantage of improved memory access, better multi-tasking and increased performance.

### When should I use OCX controls?

OCX controls come in two varieties: 16-bit and 32-bit. 16-bit controls offer compatibility with Windows and Windows for Workgroups 3.1 and 3.11. 32-bit controls work with systems running Windows NT and Windows 95. In general, you should use the most advanced version of the control that is available and is supported by your host environment.

If you are using a 32-bit programming system to develop an application that will run exclusively on a 32-bit platform, use the 32-bit OCX. If you are developing an application that must run on a mixed platform, you can use a 16-bit OCX, although you will obtain better performance if you develop separate 16-bit and 32-bit versions of your program, using the appropriate OCX controls. If you are developing exclusively for a 16-bit platform, use the 16-bit OCX.

## IsAddRow Method

Applies To

### Description

Returns whether the current row is the add row at the bottom of the grid.

### Syntax

*object* . **IsAddRow**

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the current row is the add row at the bottom of the grid..

### Remarks

This method returns a Boolean value specifying whether the current row is the add row at the bottom of the grid. This method is useful for determining if the current record is actually a new record being added.

## **IsAddRow Method Applies To**

SSDBGrid

## IsCellValid Method

[See Also](#)

[Applies To](#)

### Description

Returns whether or not the text satisfies data type validation checking.

### Syntax

*object* . **IsCellValid**

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.

### Remarks

This method returns a boolean value specifying whether or not the cell could be validated. This method can be used in the **BeforeColUpdate** event to make sure that the text/value entered can be coerced to the `DataType` of the column. It can also be used anywhere to validate an entire row prior to an application's call to `Update`. This would be particularly helpful in the **LostFocus** event.

## **IsCellValid Method Applies To**

**Column** object



## **IsCellValid See Also**

[IsTextValid](#) method

## IsItemInList Method

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Returns whether the current text in the edit portion of the combo or cell of the grid is in the dropdown list.

### Syntax

*object* . **IsItemInList**

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.

### Remarks

This method returns a boolean value that specifies whether the text in the edit portion exists in the dropdown list. **IsItemInList** causes validation of whether the text in the edit portion of the combo exists in the dropdown list.

If **ListAutoValidate = False**, the **ValidateList** event is fired and you can determine the validity.

## **IsItemInList Method Applies To**

SSDBCombo

SSDBGrid

## **IsItemInList Method See Also**

[IsTextValid](#) method

[ListAutoValidate](#) property

[ValidateList](#) event

## IsValid Method

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Returns whether or not the text satisfies validation checking.

### Syntax

*object*. **IsValid**(*ErrorCode* **As Variant**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>ErrorCode</i>	An optional parameter that will return an error code describing the error if text is invalid.

### Remarks

This method returns a Boolean value specifying whether or not the text could be validated.

## **IsValid Method Applies To**

SSDBCombo

## **IsTextValid Method See Also**

[IsCellValid](#)

[IsItemInList](#)

## Italic Property

Applies To

Example

### Description

Returns or sets the font style of the specified **Font** or **Headfont** object to either italic or non-italic.

### Syntax

*object* . **Italic**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying the font style, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	Turns on italic formatting.
<b>False</b>	(Default) Turns off italic formatting.

### Remarks

The **Font** and **Headfont** objects are not directly available at design time. At design time, you set the **Italic** property through the control's **Font** or **Headfont** property. At runtime, you can set **Italic** directly by specifying its settings for the appropriate **Font/Headfont** object.



## **Italic Property Applies To**

Font object

Headfont object

## Keyboard Interface

The following describes the keyboard interface for each of the Data Widgets controls that support keyboard use.

### SSDBGrid

<b>Press</b>	<b>To</b>	<b>Comments</b>
F4	Toggles dropdown	Only works in cells with a dropdown.
ALT + UP ARROW	Toggles dropdown	Only works in cells with a dropdown.
ALT + DOWN ARROW	Toggles dropdown	Only works in cells with a dropdown.
UP ARROW	Moves up a row in the grid	
DOWN ARROW	Moves down a row in the grid	
PAGE UP	Moves up a page in the grid	
PAGE DOWN	Moves down a page in the grid	
LEFT ARROW	Moves one cell to the left. When in edit mode, moves one character to the left.	
RIGHT ARROW	Moves one cell to the right When in edit mode, moves one character to the right.	
HOME	When in edit mode, moves to the beginning of the cell	
END	When in edit mode, moves to the end of the cell	
ESC	Restores the cell value to what it was prior to entering the cell.	
SPACE BAR	Selects the entire grid row	Only works when grid does not allow updates
TAB	Moves one cell forward	
SHIFT + TAB	Moves one cell backward	
CTRL + X	Deletes the selected row	In the case of multiple rows being selected, they will all be

DEL

Deletes the selected row

deleted.

**AllowDelete** must be set to **True**.

In the case of multiple rows being selected, they will all be deleted.

**AllowDelete** must be set to **True**.

## SSDBCombo

### Press

### To

### Comments

F4

Toggles the Data Combo's dropdown.

If the dropdown is open, it will be closed. If it is closed, it will be opened.

ALT + UP ARROW

Toggles the Data Combo's dropdown.

If the dropdown is open, it will be closed. If it is closed, it will be opened.

ALT + DOWN ARROW

Toggles the Data Combo's dropdown.

If the dropdown is open, it will be closed. If it is closed, it will be opened.

UP ARROW

Moves up a row

Only works in the dropdown portion.

DOWN ARROW

Moves down a row

Only works in the dropdown portion.

PAGE UP

Moves up a page

Only works in the dropdown portion.

PAGE DOWN

Moves down a page

Only works in the dropdown portion.

LEFT ARROW

Moves one character to the left

Works in the edit portion only.

RIGHT ARROW

Moves one character to the right

Works in the edit portion only.

HOME

Moves to the beginning of the cell

Works in the edit portion only.

END

Moves to the end of the cell

Works in the edit portion only.

ESC

When dropped down, closes the dropdown and restores the value to what it was before dropping down.

When not dropped down, restores the text to the previous database value.

ENTER

When dropped

Works only on the

down, selects the current row and closes the dropdown.

dropdown portion.

## SSDBDropDown

<b>Press</b>	<b>To</b>	<b>Comments</b>
F4	Toggles the dropdown.	Causes the dropdown to close up.
ALT + UP ARROW	Toggles the dropdown.	Causes the dropdown to close up.
ALT + DOWN ARROW	Toggles the dropdown.	Causes the dropdown to close up.
UP ARROW	Moves up a row	
DOWN ARROW	Moves down a row	
PAGE UP	Moves up a page	
PAGE DOWN	Moves down a page	
ESC	Closes the dropdown and restores the value to what it was before dropping down.	
ENTER	Selects the current row and closes up the dropdown.	

## SSDBOptSet

<b>Press</b>	<b>To</b>	<b>Comments</b>
UP ARROW	Moves up a button	
DOWN ARROW	Moves down a button	
LEFT ARROW	Moves up a button	
RIGHT ARROW	Moves down a button	
HOME	Moves to the first button in the set	
END	Moves to the last button in the set	

## LeftCol Property

[See Also](#)      [Applies To](#)

### Description

Sets or returns the left-most column in the display area of the grid.

### Syntax

*object* . **LeftCol**[= *number*]

Part	Description
------	-------------

<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
---------------	---

<i>number</i>	An integer expression specifying the left-most column.
---------------	--

**Note** To ensure that your data displays accurately, **LeftCol** should only be used when you do not have groups defined in your grid. When you do have groups, use the **LeftGrp** property. This is due to the fact that the control bases its calculation on the column for this property; if you have groups, the resulting number will be invalid.

### Remarks

The following example demonstrates LeftCol with different settings:

	Col 1	Col 2	Col 3	Col 4
▶	ACM		Association for	New York
	Bantam Books	Bantam Books Div. of:	666 Fifth Ave	New York
	Benjamin/Cummings	Benjamin-Cummings	390 Bridge Pkwy.	Redwood City
	Brady Pub.	Brady Books Div. of	15 Columerrerbus	New York
	Computer Science	Computer Science	41 Madison Ave	New York

This is **LeftCol** set to 0

	Col 3	Col 4	Col 5	Col 6
▶	Association for	New York	NY	10036
	666 Fifth Ave	New York	NY	10103
	390 Bridge Pkwy.	Redwood City	CA	94065
	15 Columerrerbus	New York	NY	10023
	41 Madison Ave	New York	NY	10010

This is **LeftCol** set to 2

## **LeftCol Property Applies To**

SSDBCombo

SSBBDropDown

SSDBGrid

**LeftCol Property See Also**

[LeftGrp](#)

## LeftGrp Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the left-most group in the display area of the grid.

### Syntax

*object* . **LeftGrp**[= *number*]

Part	Description
------	-------------

<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
---------------	---

<i>number</i>	An integer expression specifying the left-most group.
---------------	---

### Remarks

To ensure that your data displays accurately, **LeftGrp** should only be used when you have groups defined in your grid. When you do not have groups, use the **LeftCol** property. This is due to the fact that the control bases its calculation on the group for this property; if you do not have groups, the resulting number will be invalid.

The following example demonstrates LeftGrp with different settings:

Demonstration			
Group #1		Group #2	
Col #1	Col #2	Col #4	
Col #3		Col #5	Col #6
ACM		New York	
Association for Computing		NY	10036
Bantam Books	Bantam Books	New York	
666 Fifth Ave		NY	10103
Benjamin/Cummir	Benjamin-Cumr	Redwood City	
390 Bridge Pkwy.		CA	94065
Brady Pub.	Brady Books	New York	
15 Columerrrerbus Cir.errereer		NY	10023
Computer	Computer	New York	

This is **LeftGrp** set to 0



Form1

*Demonstration*

<i>Group #2</i>	
<i>Col #4</i>	
<i>Col #5</i>	<i>Col #6</i>
<b>New York</b>	
<b>NY</b>	<b>10036</b>
<b>New York</b>	
<b>NY</b>	<b>10103</b>
<b>Redwood City</b>	
<b>CA</b>	<b>94065</b>
<b>New York</b>	
<b>NY</b>	<b>10023</b>
<b>New York</b>	

This is **LeftGrp** set to 1

## **LeftGrp Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

**LeftGrp Property See Also**

[LeftCol](#)

## Level 0 (Multi-Level Row)

## **Level 1 (Multi-Level Row)**

## Level Property

Applies To

### Description

Sets or returns the column's level within a multi-level grid.

### Syntax

*object* . **Level**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the column's level in a multi-level grid.

## **Level Property Applies To**

Column object

## LevelCount Property

Applies To

### Description

Sets or returns the number of levels in a multi-level grid.

### Syntax

*object* . **LevelCount**[= *number*]

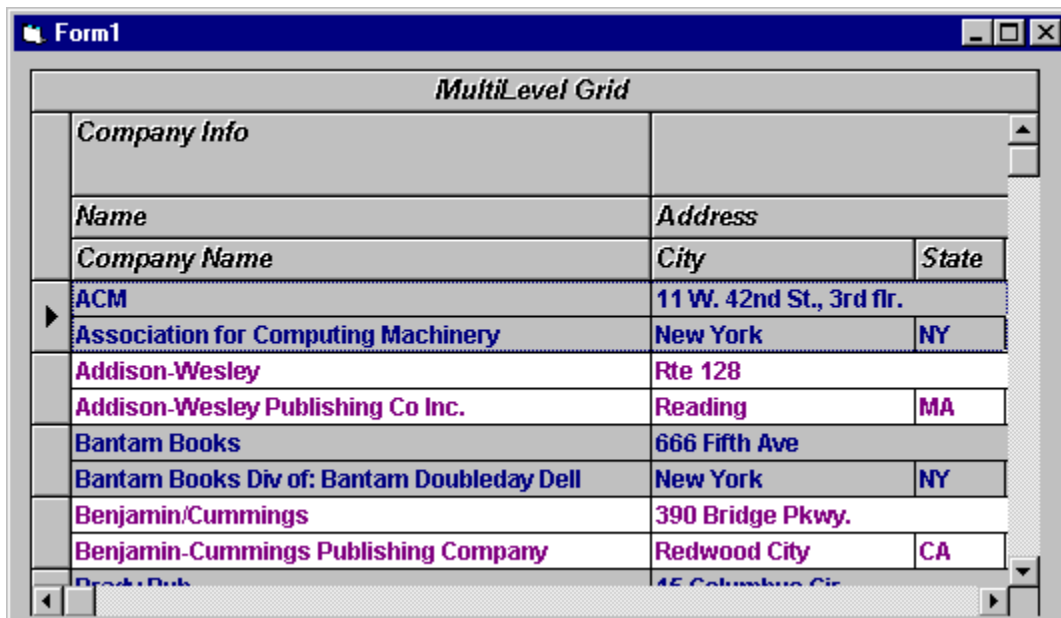
Part	Description
------	-------------

<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
---------------	---

<i>number</i>	An integer expression specifying the number of levels in a multi-level grid.
---------------	--

### Remarks

The following example demonstrates LevelCount with various settings:



In this first example, **LevelCount** = 2. You will notice that each record is displaying on two rows, specifically the **Name** and **Company Name** fields.



Form1

*MultiLevel Grid*

<i>Company Info</i>		
<i>Name</i>	<i>Address</i>	
<i>Company Name</i>	<i>City</i>	<i>State</i>
Association for Computing Machinery	New York	NY
Addison-Wesley Publishing Co Inc.	Reading	MA
Bantam Books Div of: Bantam Doubleday Dell	New York	NY
Benjamin-Cummings Publishing Company	Redwood City	CA
Brady Books Div. of Prentice Hall Pr., Simon &	New York	NY
Computer Science Press Inc Imprint of WH	New York	NY
ETN Corp.	Montoursville	PA
Gale Research, Incorporated	Detroit	MI
IEEE Computer Society Press	Los Alamitos	CA

In this first example, **LevelCount** = 1. You will notice that each record is displaying on one row.

## **LevelCount Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## List Property

Applies To

Example

### Description

Returns or sets the items contained in a combo box portion of a column with a combo box style.

### Syntax

*object* . **List** (*Index As Integer*)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Index</i>	An integer expression that specifies the index number.

### Remarks

You must first add an item using the **AddItem** method; this creates a position in the combo box. Once you have added an item, you can use the **List** property to either display or edit the value.

## List Property Applies To

Column object

## ListAutoPosition Property

[See Also](#)

[Applies To](#)

### Description

Determines whether the dropdown portion of the control will automatically position itself to the row when it is dropped down to match the value in the edit portion.

### Syntax

*object* . **ListAutoPosition**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the control should automatically position itself, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) Causes the control to automatically adjust the position of the selected row in the dropdown list when it is in its dropped state based on the current entry.
<b>False</b>	Disables this feature.

### Remarks

In cases where there is a large amount of rows in the list, you may want to consider setting this property to **False**. You can manually position the row in the list during the **PositionList** event.

## **ListAutoPosition Property Applies To**

SSDBCombo

SSDBDropDown

## **ListAutoPosition Property See Also**

[Performance Tuning](#)

## ListAutoValidate Property

Applies To

### Description

Determines whether the control will automatically check if the text entered in the edit portion is in the list.

### Syntax

*object* . **ListAutoValidate**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the text will automatically be validated, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) Causes the control to automatically search the list of values in the dropdown after the user has entered a value in the edit portion.
<b>False</b>	Disables this feature.

### Remarks

In cases where there is a large amount of rows in the list, you may want to consider setting this property to **False**. You can manually validate the row in the list during the **ValidateList** event.



## **ListAutoValidate Property Applies To**

SSDBCombo

SSDBDropDown

## ListWidth Property

[See Also](#)

[Applies To](#)

### Description

Specifies the width of the control's entire list portion.

### Syntax

*object* . **ListWidth**[= *single*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>single</i>	A number evaluating to the width of the control's list portion.

### Remarks

This width can be overridden by the [ListWidthAutoSize](#) property being set to **True**.

## **ListWidth Property Applies To**

SSDBCombo

SSBBDropDown

## **ListWidth Property See Also**

[ListWidthAutoSize](#)

## ListWidthAutoSize Property

Applies To

### Description

Determines whether the control should automatically size the dropdown based on the number of columns in the list.

### Syntax

*object* . **ListWidthAutoSize**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the control should automatically size itself, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) Causes the control to automatically size the dropdown based on the number of columns in the list.
<b>False</b>	Disables this feature.

### Remarks

The width calculated will not exceed the total width of the screen.

## **ListWidthAutoSize Property Applies To**

SSDBCombo

## Locked Property

Applies To

Example

### Description

Determines whether the column is locked from user-input.

### Syntax

*object* . **Locked**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying the status of user-input in the column, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	User input into the column is not allowed.
<b>False</b>	(Default) User input into the column is allowed.

## **Locked Property Applies To**

Column object



## MaintainBtnHeight Property

Applies To

### Description

Determines whether to keep all option buttons the same height by synchronizing the height of all buttons to the tallest one.

### Syntax

*object* . **MaintainBtnHeight**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether button height should be kept the same, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) Buttons will automatically have the same height.
<b>False</b>	Buttons will not automatically have the same height.

## **MaintainBtnHeight Property Applies To**

SSDBOptSet

## MaxDropDownItems Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the maximum number of items visible in a dropdown at once.

### Syntax

*object* . **MaxDropDownItems**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the maximum number of items visible in a dropdown at once.

## **MaxDropDownItems Property Applies To**

SSDBCombo

SSBBDropDown

**MaxDropDownItems Property See Also**

[MinDropDownItems](#)

## **MaxSelectedRows Applies To**

SSDBGrid

## MaxSelectedRows Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the maximum number of rows that can be selected at any one time.

### Syntax

*object* . **MaxSelectedRows**[= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	A long integer expression specifying the maximum number of rows that can be selected at any one time.

### Remarks

If set to 0, then there is no maximum number of rows that can be selected. The default value for this property is 100.

This property only applies when the **SelectTypeRow** property is set to "2 - Multiselect Individual" or "3 - Multiselect Range"

If the maximum number is exceeded, an error is fired in the **SelChange** event if the selection is not cancelled by the programmer.

## **MaxSelectedRows See Also**

[SelChange](#) event

[SelectTypeRow](#) property



[◀ Back](#)

## Method Summary

	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

### A

[ActiveCell](#)  
[Add](#)  
[AddItem](#)  
[AddItem \(Columns\)](#)  
[AddItemBookmark](#)  
[AddItemRowIndex](#)

### B

[ButtonFromCaption](#)  
[ButtonFromPos](#)

### C

[CellStyleSet](#)  
[CellText](#)  
[CellValue](#)  
[ColContaining](#)  
[ColPosition](#)  
[Columns](#)

### D

[DeleteSelected](#)  
[DoClick](#)

### F

[Find](#)

### G

[GetBookmark](#)  
[Groups](#)  
[GrpContaining](#)  
[GrpPosition](#)

### I

[IsAddRow](#)

IsCellValid  
IsItemInList  
IsTextValid

## **M**

MoveFirst  
MoveLast  
MoveNext  
MovePrevious  
MoveRecords

## **R**

ReBind  
Remove  
RemoveAll (Collections)  
RemoveAll (Column Object)  
RemoveAll (AddItem Mode)  
RemoveItem (Column Object)  
RemoveItem (AddItem Mode)  
RowBookmark  
RowContaining  
RowTop

## **S**

Scroll  
Soundex

## **U**

Update

## **W**

WhereIs



## MinColWidth Property

Applies To

Example

### Description

Sets or returns the minimum column width for the column containing the selected button.

### Syntax

*object* . **MinColWidth**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	A real number specifying the minimum column width.

### Remarks

The default and minimum value is 0. The maximum value is 32767.

The unit of measurement is dictated by the form's **ScaleMode** property.

## **MinColWidth Property Applies To**

SSDBOptSet

## MinDropDownItems Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the minimum number of items visible in a dropdown at once.

### Syntax

*object* . **MinDropDownItems**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the minimum number of items visible in a dropdown at once.

## **MinDropDownItems Property Applies To**

SSDBCombo

SSBBDropDown

## **MinDropDownItems Property See Also**

[MaxDropDownItems](#)



## MinHeight Property

Applies To

Example

### Description

Sets or returns the minimum height of the control.

### Syntax

*object* . **MinHeight**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	A real number specifying the minimum height.

### Remarks

The default and minimum value is 15. The maximum value is 32767.

The unit of measurement is dictated by the form's **ScaleMode** property.

## **MinHeight Property Applies To**

SSDBOptSet

## MouseIcon Property

[See Also](#)

[Applies To](#)

### Description

Sets the custom icon to be used when the mouse passes over the control.

### Syntax

*object* . **MouseIcon**[= *picture*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>picture</i>	A string expression specifying the name of a file containing a graphic, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
(None)	(Default) No Picture.
(Icon)	Determines a graphic to be used.

### Remarks

The **MousePointer** property must be set to '99' (Custom) in order to have the **MouseIcon** image display.

## **MouseIcon Property Applies To**

SSDBCombo

SSDBCommand

SSDBData

SSDBOptSet

**MouseIcon Property See Also**

[MousePointer](#)

## MousePointer Property

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Determines the icon displayed when the mouse passes over the control.

### Syntax

*object* . **MousePointer**[= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying what image to use, as described in Settings.

### Settings

Setting	Description
0	(Default) Default value
1	Arrow
2	Cross
3	I-Beam
4	Icon
5	Size
6	Size NE SW
7	Size N S
8	Size NW SE
9	Size W E
10	Up Arrow
11	Hourglass
12	NoDrop
13	Arrow and Hourglass
14	Arrow and Question
15	Size All
99	Custom

There are [constants](#) available for the settings of this property.

## **MousePointer Property Applies To**

SSDBCombo

SSDBCommand

SSDBData

SSDBOptSet

**MousePointer Property See Also**

[MouseIcon](#)



## MoveFirst Method

[See Also](#)

[Applies To](#)

### Description

Moves to the first record in the grid.

### Syntax

*object* . **MoveFirst**

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.

## **MoveFirst Method Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **MoveFirst Method See Also**

[MoveLast](#)

[MoveNext](#)

[MovePrevious](#)

[MoveRecords](#)

## MoveLast Method

[See Also](#)

[Applies To](#)

### Description

Moves to the last record in the grid.

### Syntax

*object* . **MoveLast**

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.

## **MoveLast Method Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **MoveLast Method See Also**

[MoveFirst](#)

[MoveNext](#)

[MovePrevious](#)

[MoveRecords](#)

## MoveNext Method

[See Also](#)

[Applies To](#)

### Description

Moves to the next record in the grid.

### Syntax

*object* . **MoveNext**

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.

## **MoveNext Method Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid



## **MoveNext Method See Also**

[MoveFirst](#)

[MoveLast](#)

[MovePrevious](#)

[MoveRecords](#)

## MovePrevious Method

[See Also](#)

[Applies To](#)

### Description

Moves to the previous record in the grid.

### Syntax

*object* . **MovePrevious**

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.

## **MovePrevious Method Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **MovePrevious Method See Also**

[MoveFirst](#)

[MoveLast](#)

[MoveNext](#)

[MoveRecords](#)

## MoveRecords Method

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Moves a specified number of records in the grid.

### Syntax

*object* . **MoveRecords**( *Records* **As Long**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Records</i>	A long integer specifying the number of records in the grid to move.

### Remarks

To move forward in the grid, specify a positive number of records. If the number of records to move exceeds the actual length of the grid, the last record will be selected.

To move backwards in the grid, specify a negative number of records. If the number of records to move exceeds the actual length of the grid, the first record will be selected.

## **MoveRecords Method Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **MoveRecords Method See Also**

[MoveFirst](#)

[MoveLast](#)

[MoveNext](#)

[MovePrevious](#)

## **MultiLine Property Applies To**

SSDBCombo

SSDBGrid



## **MultiLine Property See Also**

[WordWrap](#)

## MultiLine Property

[See Also](#)

[Applies To](#)

### Description

Determines whether the control will display multiple lines of text.

### Syntax

*object* . **MultiLine**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression how text will be displayed, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) The object will display multiple lines of text.
<b>False</b>	Text for the object will be displayed on a single line.

### Remarks

When applied to the DataGrid, the MultiLine property determines whether multiple lines of text will be displayed in the cells of the grid.

When Applied to the DataCombo, MultiLine determines whether multiple lines will be displayed in the edit portion of the combo.

## Name Property

Applies To

### Description

For the **Font** and **HeadFont** objects, returns or sets the font name.

For the **StyleSet** object, returns or sets the name of the StyleSet.

For the **Column** object, returns or sets the column name.

### Syntax

*object* . **Name**[= *font* ]

*object* . **Name**[= *stylesheet* ]

*object* . **Name**[= *string* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>font</i>	A string expression specifying the name of the font to use.
<i>stylesheet</i>	A string expression specifying the name of the stylesheet to use.
<i>string</i>	A string expression specifying the name of the column.

### Remarks

The **Font** and **Headfont** objects are not directly available at design time. At design time, set the **Name** property through the control's **Font** or **Headfont** property. At runtime, you can set **Name** directly by specifying its setting for the appropriate **Font/Headfont** object.

The **Column** object uses the name property as a unique identifier. The DataGrid control will prevent the creation of duplicate names for column objects. The **Name** property is independent of the **Caption** property for the object, but when referencing an object using a string value, the control will first search through the column captions for a matching string before searching the column names.

The **Name** property of the Column object primarily provides you with a way to create columns with duplicate captions in a DataGrid that is in Unbound or AddItem mode.

## **Name Property Applies To**

Font object

Headfont object

StyleSet object

### **Note on the control-level Bookmark property**

This does not apply to the **Bookmark** property of the ssRowBuffer object.

The Bookmark property at the control level is a standard property that is returned as a byte array. Some environments may treat the returned value as an integer, but that behavior is not standard and should not be relied upon.

Note that when using a control in AddItem mode, the value returned by **Bookmark** is not equivalent to the absolute row number. To obtain the row number, use the **AddItemRowIndex** method.

## Nullable Property

Applies To

### Description

Determines how the control stores null or empty data in the database.

### Syntax

*object* . **Nullable**[= *number* ]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression that specifies how null values will be stored, as described in Settings.

### Settings

Setting	Description
0	Automatic. If a field contains a null value or an empty string, the control will first check the data source to see if null values are allowed. If nulls are allowed, a null value will be stored in the database. Otherwise an empty string will be stored.
1	(Default) Null. The data will be stored as a null value.
2	Empty String. The data will be stored as an empty string value ("").

### Remarks

Different databases deal with null values in different ways. Since the Data Widgets controls are designed to work with a variety of data sources, the controls have the ability to query the back end and find out which way to store null values. Depending on the type of connection to the database, this can have a significant impact on performance.

**Note** If the database does not support null values, and you attempt to store nulls by setting **Nullable** to 1, an error will result.

If you know how the database handles the storage of null values, you can improve performance by setting the **Nullable** property to either 1 or 2. Setting this value to 0 will provide a greater range of compatibility, but performance will suffer.

If you encounter problems when you attempt to save a record that contains a null value, you can change the setting of **Nullable**, which should fix the problem. In any case, you should implement error-checking code to insure that the storage operation succeeded.

**Note** The setting of this property controls how the Data Widgets control will attempt to store the null value. In some cases, the data control or the database back end may change the null value before actually committing it to the database.

There are constants available for the settings of this property.

## **Nullable Property Applies To**

Column object

## NumberFormat Property

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Sets or returns the display format for the object.

### Syntax

*object* . **NumberFormat**[= *format*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>format</i>	A string expression that evaluates to the format string for the object.

### Remarks

The **NumberFormat** property makes use of the same formatting strings as Visual Basic's **Format** function. These may be named format strings or user-defined format strings.

The following is a list of the named format strings that can be used with **NumberFormat**:

Format name	Description
<b>General Number</b>	Display number as is, with no thousand separators.
<b>Currency</b>	Display number with thousand separator, if appropriate; display two digits to the right of the decimal separator. Note that output is based on system locale settings.
<b>Fixed</b>	Display at least one digit to the left and two digits to the right of the decimal separator.
<b>Standard</b>	Display number with thousands separator, at least one digit to the left and two digits to the right of the decimal separator.
<b>Percent</b>	Display number multiplied by 100 with a percent sign (%) appended to the right; always display two digits to the right of the decimal separator.
<b>Scientific</b>	Use standard scientific notation.
<b>Yes/No</b>	Display No if number is 0; otherwise, display Yes.
<b>True/False</b>	Display False if number is 0; otherwise, display True.
<b>On/Off</b>	Display Off if number is 0; otherwise, display On.

Use the following as a guide to creating user-defined format strings for use with the **NumberFormat** property:

Character	Description
<b>None</b>	<b>No formatting.</b> Display the number with no formatting.
<b>0</b>	<b>Digit placeholder.</b> Display a digit or a zero. If the expression has a digit in the position where the 0 appears in the format string, display it; otherwise, display a zero in that position.



If the number has fewer digits than there are zeros (on either side of the decimal) in the format expression, display leading or trailing zeros. If the number has more digits to the right of the decimal separator than there are zeros to the right of the decimal separator in the format expression, round the number to as many decimal places as there are zeros. If the number has more digits to the left of the decimal separator than there are zeros to the left of the decimal separator in the format expression, display the extra digits without modification.

**#** **Digit placeholder.** Display a digit or nothing. If the expression has a digit in the position where the # appears in the format string, display it; otherwise, display nothing in that position.

This symbol works like the 0 digit placeholder, except that leading and trailing zeros aren't displayed if the number has the same or fewer digits than there are # characters on either side of the decimal separator in the format expression.

**.** **Decimal placeholder.** In some locales, a comma is used as the decimal separator. The decimal placeholder determines how many digits are displayed to the left and right of the decimal separator. If the format expression contains only number signs to the left of this symbol, numbers smaller than 1 begin with a decimal separator. If you always want a leading zero displayed with fractional numbers, use 0 as the first digit placeholder to the left of the decimal separator instead. The actual character used as a decimal placeholder in the formatted output depends on the Number Format recognized by your system.

**%** **Percentage placeholder.** The expression is multiplied by 100. The percent character (%) is inserted in the position where it appears in the format string.

**,** **Thousand separator.** In some locales, a period is used as a thousand separator. The thousand separator separates thousands from hundreds within a number that has four or more places to the left of the decimal separator. Standard use of the thousand separator is specified if the format contains a thousand separator surrounded by digit placeholders (0 or #). Two adjacent thousand separators or a thousand separator immediately to the left of the decimal separator (whether or not a decimal is specified) means "scale the number by dividing it by 1000, rounding as needed."

You can scale large numbers using this technique. For example, you can use the format string "# #0,," to represent 100 million as 100. Numbers smaller than 1 million are displayed as 0. Two adjacent thousand separators in any position other than immediately to the left of the decimal separator are treated simply as specifying the use of a thousand separator. The actual character used as the thousand separator in the formatted output depends on the Number Format recognized by your system.

**:** **Time separator.** In some locales, other characters may be used to

represent the time separator. The time separator separates hours, minutes, and seconds when time values are formatted. The actual character used as the time separator in formatted output is determined by your system settings.

**/** **Date separator.** In some locales, other characters may be used to represent the date separator. The date separator separates the day, month, and year when date values are formatted. The actual character used as the date separator in formatted output is determined by your system settings.

**E- E+ e- e+** **Scientific format.** If the format expression contains at least one digit placeholder (0 or #) to the right of E-, E+, e-, or e+, the number is displayed in scientific format and E or e is inserted between the number and its exponent. The number of digit placeholders to the right determines the number of digits in the exponent. Use E- or e- to place a minus sign next to negative exponents. Use E+ or e+ to place a minus sign next to negative exponents and a plus sign next to positive exponents.

**- + \$ ( ) space** **Literal character.** Displays the character specified. To display a character other than one of those listed, precede it with a backslash (\) or enclose it in double quotation marks (" ").

**\** Display the next character in the format string as a literal character. Many characters in the format expression have a special meaning and can't be displayed as literal characters unless they are preceded by a backslash. The backslash itself isn't displayed. Using a backslash is the same as enclosing the next character in double quotation marks. To display a backslash, use two backslashes (\\).  
Examples of characters that can't be displayed as literal characters are the date- and time-formatting characters (a, c, d, h, m, n, p, q, s, t, w, y, and /:), the numeric-formatting characters (#, 0, %, E, e, comma, and period), and the string-formatting characters (@, &, <, >, and !).

**"ABC"** Display the string inside the double quotation marks. To include a string in format from within code, you must use Chr(34) to enclose the text (34 is the character code for a double quotation mark).

## **NumberFormat Property Applies To**

Column object

## **NumberFormat Property See Also**

Visual Basic's **Format** Function

## NumberOfButtons Property

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Sets or returns the number of option buttons in the DataOptionSet.

### Syntax

*object* . **NumberOfButtons**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the number of buttons in the DataOptionSet.

### Remarks

Valid range is 1 to 100 with 1 as the default at design time and 0 at runtime.

## **NumberOfButtons Property Applies To**

SSDBOptSet

## **NumberOfButtons Property See Also**

[IndexSelected](#)

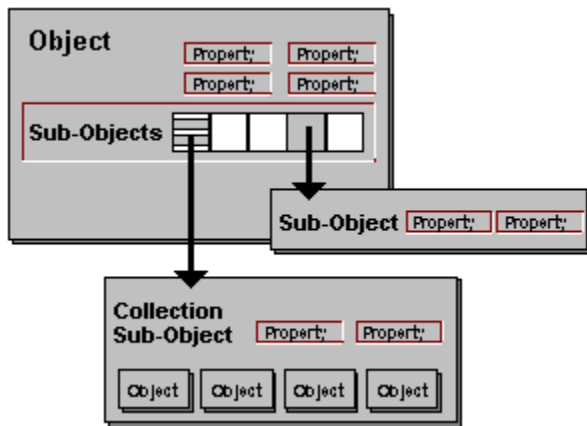
## Object Concepts

This section will be of special interest to programmers who have worked with earlier versions of our custom controls. It highlights the major differences between the older controls you may be familiar with and the newer controls you now have.

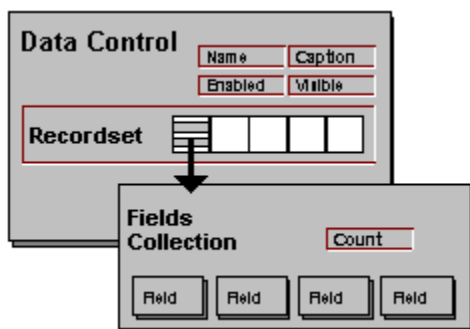
Object-oriented programming offers you greater power than before, with less work on your part. However, because this is a new technology, there are some new concepts with which you should be familiar. This section provides a brief introduction to some of the new concepts you will encounter while using Sheridan custom controls.

## Sub-objects and Collections

Data Widgets provides an object-oriented approach to programming through the use of *sub-objects* and *collections*. An *object* refers to a single unit or entity within your application which contains both code and data. Objects can contain other objects, which have properties and methods of their own that can be examined and changed. Objects may also contain *collection* objects. A collection is a special type of object that contains sub-objects that are all of the same type, or *class*.



You are probably familiar with the concept of sub-objects if you have used the Visual Basic data control. The **Recordset** object is a sub-object of the Visual Basic Data Control. The **Recordset** contains a collection sub-object called the **Fields** object, which contains information that relates to all the fields in the **Recordset** collectively. The **Fields** collection also contains the **Field** objects themselves, which store data and also information pertaining to that data.





Objects within collections often have this type of "paired" arrangement; a single collection object (**Fields**) which describes and contains the collection as a whole, and multiple member objects (**Field**) which make up the collection. In addition, there is usually a corresponding property of the same name as the object that returns information about the object.

Collections have replaced property arrays as the preferred method for accessing sets of controls at runtime. This means you no longer have to specify an array for each property you wish to access, and there are fewer special property names. For example, previously to set the alignment of the fifth column in a DataGrid control, you would have used the following code:

```
SSDBGrid1.ColAlignment(4) = 0 'Left aligned
```

Now, you would use the standard **Alignment** property, specifying instead the object in the collection to which it will apply:

```
SSDBGrid1.Columns(4).Alignment = 0 'Left Aligned
```

This makes it especially easy to apply multiple properties to an object using the new `With.. End With` statements in Visual Basic 4.0:

```
With SSDBGrid1.Columns(4)
    .Alignment = 0
    .BackColor = vbRed
    .ForeColor = vbWhite
    .Caption = "Column 5"
EndWith
```

This control's object type is:

SSDBCombo

This control's object type is:  
SSDBCommand

This control's object type is:  
SSDBDropDown

This control's object type is:  
SSDBGrid

This control's object type is:  
SSDBOptSet

This control's object type is:

SSDBData

[◀ Back](#)

## Object Summary

[ActiveCell](#)

[Bookmark](#)

[Button](#)

[Column](#)

[Font](#)

[Group](#)

[HeadFont](#)

[ssRowBuffer](#)

[StyleSet](#)



**Odd Row (Row 1)**

## Optimizing Data Widgets

### Improving Load Time

By default, the Data Grid, Data Combo, and Data DropDown custom controls each go to the last record in a record set to determine the exact number of rows. The controls do this to give an accurate number of rows in the **Rows** property. However, with large databases, this could cause a decrease in performance.

To turn this option off, set the **UseExactRowCount** property to **False**. This will cause the control to estimate the number of rows in the record set. If this property is set to **False**, *do not rely on the **Rows** property for an accurate number of rows.* If you do a **MoveLast** on the data control's record set, the **Rows** property will be accurate.

### Optimizing the Data Combo and Data DropDown

The Data Combo and Data DropDown can be optimized when performing certain functions. There are two functions that the controls perform automatically which can be overridden.

#### Auto List Validation

The Data Combo and Data DropDown automatically perform validation of the value in the edit portion of the Data Combo or the cell of a Data Grid against the values in the list portion to find a match. In some circumstances, this validation can be very slow, since the control must sequentially search the entire database. With large databases, this operation can be quite slow. To turn this feature off and perform your own validation of the field, set the **ListAutoValidate** property to **False**. This will cause the control to skip the validation process and trigger the **ValidateList** event.

#### Auto Positioning

Another specific way of optimizing the performance of the Data Combo or Data DropDown controls is to set the **ListAutoPosition** property to **False**. This turns off the automatic positioning of the list based on the contents of the edit portion of the Data Combo or cell of the Data Grid. Instead, the **PositionList** event will be triggered.

Similar to the validation of data against the list, the positioning requires the control to search the list sequentially which can cause a performance penalty with large databases.

## OptionValue Property

[See Also](#)

[Applies To](#)

[Example](#)

### Description

The value that is compared against or given to the bound field. When reading from a database, if the two values are equal, then the button is selected. When writing to the database, the field will receive the value indicated in the **OptionValue** property.

### Syntax

*object* . **OptionValue**[= *text*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Text</i>	A string expression that evaluates to the value of a field.

### Remarks

Each button must have unique option values.

## **OptionValue Property Applies To**

Button object

SSDBOptSet

## **OptionValue Property See Also**

[DataField](#)

[DataSource](#)

## Orientation Property

Applies To

### Description

Sets or returns the control's display orientation, either horizontal or vertical.

### Syntax

*object* . **Orientation**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the display orientation of the control, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
0	(Default) The control will be displayed horizontally.
1	The control will be displayed vertically.

There are constants available for the settings of this property.

### Remarks

The **Align** property will automatically change the state of the **Orientation** property as needed if these properties do not coincide.

## **Orientation Property Applies To**

SSDBData

## **Visual C++ Examples**

[ActiveCell Object example](#)

[Create StyleSet example](#)

[Create StyleSet with Wrapper class example](#)

[Get SelBookmarks with Wrapper](#)

**See Also:** [Answers to Common Questions](#)



## PageValue Property

Applies To

Example

### Description

Determines the number of rows the control will move forward or backward in the record set when the user moves to the next or previous page.

### Syntax

*object* . **PageValue**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the number of rows.

### Remarks

**SSDBCommand:** Valid range is 2-1000. The default value is 20.

**SSDBData:** Valid range is 1-300. The default value is 20.

## **PageValue Property Applies To**

SSDBCommand

SSDBData

## Performance Tuning

Data Widgets 2.0 provides a number of ways for you to improve the performance of your applications. In addition to the following suggestions, you may also be able to increase the performance of controls operating in Bound mode by changing the settings of your database engine. Consult the documentation for your development environment for information on how to do this.

### Bound Mode Performance Tuning

When using the Data Combo, Data DropDown or DataGrid in Bound mode, you can set the **UseExactRowCount** property to False to increase performance. However, doing so will reduce the accuracy of the control in estimating the true size of the record set. This primarily effects the scroll bar; the size and placement of the "thumb" will be less accurate when **UseExactRowCount** is set to False.

When using the Data Combo or Data DropDown, setting **ListAutoPosition** to False will further speed up the operation of the control, as it will not have to perform a search until the full text string is entered. When **ListAutoPosition** is True, the control will perform a search as each keystroke is entered. See [below](#) for more information on using **ListAutoPosition** in Unbound mode.

Some data sources, particularly remote data sources, may respond slowly when queried for information. The Data Widgets controls sometimes request information from the data source "behind the scenes" in order to perform routine functions. Disabling requests for default information can further improve performance. You may want to try different settings for the **Nullable** and **UseDefaults** properties to see if you can gain an improvement when using remote data sources.

### Unbound Mode Performance Tuning

Performance can be enhanced in Unbound mode through use of the **ReadType** property of the **ssRowBuffer** object. You can check the value of this property during an **UnboundReadData** event and use it to optimize the code you are using to perform the read. Refer to the [example code](#) for the **ReadType** property for a complete illustration of this concept.

Often, a control in Unbound mode does not require all the information from your data source. For example, when you are repositioning the data in a DataGrid by dragging the thumb on the scroll bar, the control does not need to read each column of data for the records you are passing through. A bookmark alone is sufficient to tell the control whether it has reached the correct position in the data set. If your code is returning the information from each field during this operation, that information is read and discarded, seriously degrading performance. By checking the value of **ReadType**, you can determine what type of operation the control is performing, and return only the data required, in this case the bookmark for each row.

Another situation in which correct use of the **ReadType** property can really speed up your application is when using the DataCombo or the DataDropDown in Unbound mode with the **ListAutoPosition** property set to True. In this mode, the control will attempt to position itself according to what the user types in the edit portion of the control, effectively executing a new search with each keystroke. Again, if your code returns the values for every column in the control's data source, your application will spend extra time retrieving unneeded data. The only data required in this case is the bookmark of each row and the value of the single column to which the control is bound. This is the only field used in the search, so other data would be superfluous. By examining the setting of **ReadType** before performing the read, your **UnboundReadData** code will spend its time retrieving only the

required data.

To optimize scrolling performance of controls in Unbound mode, use the **UnboundPositionData** event. This event allows you to specify a particular location in the data set to begin the display of records. This eliminates the need to read through all the rows between the current row and the new position. For example, if the grid is currently at row 5, and the user scrolls to row 7000, you can use the **UnboundPositionData** event to bypass reading rows 6 to 6999, thus eliminating approximately 700 calls to the **UnboundReadData** event.

If you know the size of the record set you are using in Unbound mode, you can set the **Rows** property of the control to the number of records. This allows the control to optimize the number of rows that must be read at one time.

## **AddItem Mode Performance Tuning**

To improve the performance of a control functioning in AddItem mode, be sure to set the **Redraw** property to False before adding a block of items. You must then set **Redraw** to True once you have finished adding the data. This provides two performance benefits. First, it disables repainting of the control while new items are being added, eliminating the graphics overhead of drawing each entry (and display flicker which might distract the user.) Second, setting the **Redraw** property sets a flag inside the control which determines how data caching is handled. With **Redraw** set to False, data caching is optimized for block additions, further enhancing control performance.

[◀ Back](#)

## **Performing calculations on the values in a DataGrid**

### **How would you like the totals to be displayed?**

Display column totals in a separate control (i.e. a text box)

Display computed values from a row in their own column

## Picture Property

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Determines a picture object for display.

### Syntax

*object* . **Picture**[= *picture*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>picture</i>	A string expression specifying the name of a file containing a graphic, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
(None)	(Default) No Picture.
(Bitmap, Icon, Metafile)	Determines a graphic. You can load the graphic from the properties window at design time. At runtime, you can set this property using the <b>LoadPicture</b> function on a bitmap, icon, or metafile.

### Remarks

This property will set the picture for the specified object. If the picture is a Windows metafile, you must also set the height and width used to display the picture via the appropriate [PictureMetaHeight](#) and [PictureMetaWidth](#) properties for this object.

## **Picture Property Applies To**

Button object

SSDBCommand

SSDBOptSet

## **Picture Property See Also**

[AutoSize](#)

[PictureMetaHeight](#)

[PictureMetaWidth](#)

[PictureAlignment](#)



## PictureAlignment Property

[See Also](#)

[Applies To](#)

### Description

Determines the alignment of the graphic specified in the **Picture** property.

### Syntax

*object* . **PictureAlignment**[= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the alignment of the picture as described in Settings.

### Settings (SSDBCommand)

Setting	Description
0	(Default) Left justify - Top
1	Left justify - Middle
2	Left justify - Bottom
3	Right justify - Top
4	Right justify - Middle
5	Right justify - Bottom
6	Centered - Top
7	Centered - Middle
8	Centered - Bottom
9	Left of caption
10	Right of caption
11	Above caption
12	Below caption
13	Stretch
14	Tile

There are [constants](#) available for the settings of this property.

### Settings (SSDBOptSet)

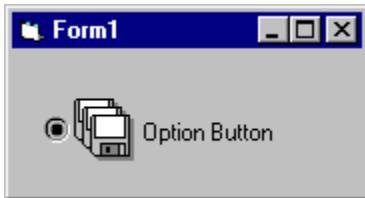
Setting	Description
0	Left of text
1	(Default) Right of text
2	Fit to caption
3	Tile

There are constants available for the settings of this property.

### Remarks

For the SSDBOptSet control, this property determines placement of the picture relative to the option button. This property is control-specific, and affects all buttons within the SSDBOptSet control.

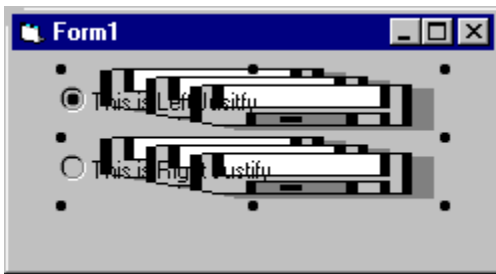
The following examples demonstrate the various alignment positions as it applies to the SSDBOptSet control:



This is left of text



This is right of text



This is fit to caption



This is tiled

## **PictureAlignment Property Applies To**

SSDBCommand

SSDBOptSet

**PictureAlignment Property See Also**

[CaptionAlignment](#)

## PictureButton Property

[See Also](#)

[Applies To](#)

### Description

Determines the picture to be used for the default button.

### Syntax

*object* . **PictureButton**[= *picture*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>picture</i>	A string expression specifying the name of a file containing a graphic, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
(None)	(Default) No Picture.
(Bitmap)	Determines a graphic. You can load the graphic from the properties window at design time. At runtime, you can set this property using the <b>LoadPicture</b> function on a bitmap.

### Remarks

This property will set the picture for the button (an ellipsis by default).

## **PictureBox Property Applies To**

SSDBGrid

## **PictureButton Property See Also**

[PictureComboButton](#)

[PictureRecordSelectors](#)

## PictureButtons Property

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Determines the file containing the pictures to be used for each button in the control.

### Syntax

*object* . **PictureButtons**[= *picture*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Picture</i>	A string expression specifying the name of a file containing a graphic, as described in Settings.

### Settings

Setting	Description
(None)	(Default) No Picture.
(Bitmap)	Determines a graphic containing the pictures to be used for each button in the control.

### Remarks

The default bitmaps used for the picture buttons are built into the control. A bitmap file is supplied to use as a guide when creating custom bitmaps.

Bitmaps for all buttons must be included in the file even if the button will be turned off. Buttons can be any size, divided into equal parts.

SSDBData requires a bitmap be used. When this property is set to "None", the default bitmap built into the control will be used. If you have selected an external bitmap, "Bitmap" will appear in the properties list.



## **PictureButtons Property Applies To**

SSDBData

## **PictureButtons Property See Also**

[PictureCaption](#)

[PictureCaptionAlignment](#)

## PictureCaption Property

[See Also](#)

[Applies To](#)

### Description

Determines the picture to be drawn in the caption section of the control.

### Syntax

*object* . **PictureCaption**[= *picture*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>picture</i>	A string expression specifying the name of a file containing a graphic, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
(None)	(Default) No Picture.
(Bitmap, Icon, Metafile)	Determines a graphic for the caption area of the control.

### Remarks

If you wish to set this property at runtime, you must use the syntax:

```
PictureCaption = LoadPicture("filename")
```

## **PictureCaption Property Applies To**

SSDBData

## **PictureCaption Property See Also**

[PictureButtons](#)

[PictureCaptionAlignment](#)

## PictureCaptionAlignment Property

[See Also](#)

[Applies To](#)

### Description

Determines how the caption picture will be aligned within the caption section of the control relative to the caption text.

### Syntax

*object* . **PictureCaptionAlignment**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying which display state to use, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
0	Left - Top
1	Left - Middle
2	Left - Bottom
3	Right - Top
4	(Default ) Right - Middle
5	Right - Bottom
6	Center - Top
7	Center - Middle
8	Center - Bottom
9	Left of Caption
10	Right of Caption
11	Above Caption
12	Below Caption
13	Fit to Caption
14	Tile

There are [constants](#) available for the settings of this property.

## **PictureCaptionAlignment Property Applies To**

SSDBData

## **PictureCaptionAlignment Property See Also**

[PictureButtons](#)

[PictureCaption](#)



## PictureCaptionMetaHeight

Applies To

### Description

Sets or returns the height of a picture if it is a metafile.

### Syntax

*object* . **PictureCaptionMetaHeight**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression that evaluates to the height of a metafile selected as a picture.

### Remarks

This property affects the PictureCaption of the Enhanced Data Control.

## **PictureCaptionMetaHeight Applies To**

SSDBData

## PictureCaptionMetaWidth

Applies To

### Description

Sets or returns the width of a picture if it is a metafile.

### Syntax

*object* . **PictureCaptionMetaWidth**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression that evaluates to the width of a metafile selected as a picture.

### Remarks

This property affects the PictureCaption of the Enhanced Data Control.

## **PictureCaptionMetaWidth Applies To**

SSDBData

## PictureComboButton Property

[See Also](#)

[Applies To](#)

### Description

Determines the picture to be used for the dropdown button.

### Syntax

*object* . **PictureComboButton**[= *picture*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>picture</i>	A string expression specifying the name of a file containing a graphic, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
(None)	(Default) No Picture.
(Bitmap)	Determines a graphic. You can load the graphic from the properties window at design time. At runtime, you can set this property using the <b>LoadPicture</b> function on a bitmap.

## **PictureComboButton Property Applies To**

SSDBGrid

## **PictureComboButton Property See Also**

[PictureButton](#)

[PictureRecordSelectors](#)

## PictureDropDown Property

Applies To

### Description

Returns or sets a **Picture** object for the picture that will appear on the dropdown button in place of the down arrow.

### Syntax

*object* . **PictureDropDown**[= *picture*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>picture</i>	An expression specifying a graphic, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
(None)	(Default) No Picture.
(Bitmap)	Determines a graphic. You can load the graphic from the properties window at design time. At runtime, you can set this property using the <b>LoadPicture</b> function.



## **PictureDropDown Property Applies To**

SSDBCombo

## PictureMetaHeight Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the height of a picture if it is a metafile.

### Syntax

*object* . **PictureMetaHeight**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression that evaluates to the height of a metafile selected as a picture.

## **PictureMetaHeight Property Applies To**

Button object

SSDBCommand

SSDBOptSet

## **PictureMetaHeight Property See Also**

[Picture](#)

[PictureMetaWidth](#)

## PictureMetaWidth Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the width of a picture if it is a metafile.

### Syntax

*object* . **PictureMetaWidth**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression that evaluates to the width of a metafile selected as a picture.

## **PictureMetaWidth Property Applies To**

Button object

SSDBCommand

SSDBOptSet

## **PictureMetaWidth Property See Also**

[Picture](#)

[PictureMetaHeight](#)

## PictureRecordSelectors Property

[See Also](#)

[Applies To](#)

### Description

Determines the segmented bitmap for the record selectors graphic.

### Syntax

*object* . **PictureRecordSelectors**[= *picture*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>picture</i>	A string expression specifying the name of a file containing a graphic, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
(None)	(Default) No Picture.
(Bitmap)	Determines a graphic. You can load the graphic from the properties window at design time. At runtime, you can set this property using the <b>LoadPicture</b> function on a bitmap.

### Remarks

By default, the record selectors are shown as an arrow, pencil, or an asterisk. This property allows you to specify your own segmented bitmap showing all three states (active record, dirty record, new record).

The control will automatically split the graphic in thirds, therefore all three images must be equal in size, and in the order of the state you wish to use (active, dirty, and new).



## **PictureRecordSelectors Property Applies To**

SSDBGrid

## **PictureRecordSelectors Property See Also**

[PictureButton](#)

[PictureComboButton](#)

## Position Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the visible position of the column or group within the grid.

### Syntax

*object* . **Position**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the position of the object.

### Remarks

The position number is relative to the entire grid. The **Position** of a given column does not reflect group assignments.

**Position** is a 1-based property. The leftmost column that is visible in the grid has a **Position** value of one. The next column to the right has a **Position** of two, regardless of whether it is in the same group as the first column.

When the grid is scrolled horizontally, **Position** is updated dynamically as columns and groups move across the visible area of the control. Also, if columns are re-arranged within the grid, either through code or via drag-and-drop, their **Position** changes accordingly.

## **Position Property Applies To**

Column object

Group object

## **Position Property See Also**

ColPosition method

GrpPosition method

## PositionList Event

See Also

Applies To

### Description

Occurs when the control needs to position the dropdown list to match the contents of the edit portion or for SSDBDropDown, the cell value in the Data Grid.

### Syntax

**Sub** control\_ **PositionList** (*Text* **As String**)

<b>Part</b>	<b>Description</b>
-------------	--------------------

---

<i>Text</i>	The text to match in the list of values
-------------	---

### Remarks

If the control has the **ListAutoPosition** property set to **False**, the control will trigger this event when it needs to position the list so that the current edit text appears in the visible portion of the list.

When this event occurs, the *Text* parameter will contain the value to search for in the list.

You should set the **ListAutoPosition** property to **False** and process this event to optimize list positioning in large sets. Since the control cannot position a data control using index fields due to a Visual Basic limitation, using the data control 'Find' methods in this event can significantly improve positioning time.

## **PositionList Event Applies To**

SSDBCombo

SSBBDropDown

## **PositionList Event See Also**

[ListAutoPosition](#)



[◀ Back](#)

## Property Summary

	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

[\(About\)](#)

[\(Custom\)](#)

### A

[ActiveRowStyleSet](#)

[Alignment \(Column Object\)](#)

[Alignment \(SSDBData\)](#)

[AlignmentPicture](#)

[AlignmentText](#)

[AllowAddNew](#)

[AllowColumnMoving](#)

[AllowColumnShrinking](#)

[AllowColumnSizing](#)

[AllowColumnSwapping](#)

[AllowDelete](#)

[AllowDragDrop](#)

[AllowGroupMoving](#)

[AllowGroupShrinking](#)

[AllowGroupSizing](#)

[AllowGroupSwapping](#)

[AllowInput](#)

[AllowNull](#)

[AllowRowSizing](#)

[AllowSizing](#)

[AllowUpdate](#)

[AutoRestore](#)

[AutoSize](#)

### B

[BackColor](#)

[BackColorEven](#)

[BackColorOdd](#)

[BalloonHelp](#)

[BevelColorFace](#)

[BevelColorFrame](#)

[BevelColorHighlight](#)

BevelColorScheme  
BevelColorShadow  
BevelInner  
BevelOuter  
BevelType  
BevelWidth  
Bold  
Bookmark (ssRowBuffer)  
BookmarkDisplay  
BookmarksToKeep  
BorderStyle  
BorderWidth  
ButtonEnabled  
ButtonsAlways  
ButtonSize  
ButtonVisible

## C

Caption  
CaptionAlignment  
Case  
CellNavigation  
CheckBox3D  
Col  
ColChanged  
ColOffset  
ColorMask  
ColorMaskEnabled  
Cols  
ColumnHeaders  
ColWidth  
ComboDroppedDown  
Count

## D

DatabaseAction  
DataField  
DataFieldList  
DataFieldToDisplay  
DataMode  
DataSourceList

Data Type  
DefColWidth  
DelayInitial  
DelaySubsequent  
DividerStyle  
DividerType  
DroppedDown

## F

FieldDelimiter  
FieldLen  
FieldSeparator  
FindBufferSize  
FindDialog  
FindFieldExclude  
FindFieldInclude  
FirstRow  
Font3D  
ForeColorEven  
ForeColorOdd

## G

GroupHeaders  
GroupHeadLines

## H

HasBackColor  
HasForeColor  
HasHeadBackColor  
HasHeadForeColor  
HeadBackColor  
HeadFont3D  
HeadForeColor  
HeadLines  
HeadStyleSet  
HeightGap  
hWndEdit

## I

IndexSelected  
Italic

## L

LeftCol

LeftGrp

Level

LevelCount

List

ListAutoPosition

ListAutoValidate

ListWidth

ListWidthAutoSize

Locked

## M

MaintainBtnHeight

MaxDropDownItems

MaxSelectedRows

MinColWidth

MinDropDownItems

MinHeight

MouseIcon

MousePointer

MultiLine

## N

Name

Nullable

NumberFormat

NumberOfButtons

## O

OptionValue

Orientation

## P

PageValue

Picture

PictureAlignment

PictureButton

PictureButtons

PictureCaption

[PictureCaptionAlignment](#)  
[PictureCaptionMetaHeight](#)  
[PictureCaptionMetaWidth](#)  
[PictureComboButton](#)  
[PictureDropDown](#)  
[PictureMetaHeight](#)  
[PictureMetaWidth](#)  
[PictureRecordSelectors](#)  
[Position](#)

## R

[ReadType](#)  
[RecordSelectors](#)  
[Redraw](#)  
[ResizeHeight](#)  
[ResizeWidth](#)  
[RotateText](#)  
[RoundedCorners](#)  
[Row](#)  
[RowChanged](#)  
[RowCount](#)  
[RowHeight](#)  
[RowNavigation](#)  
[RowOffset](#)  
[RowSelectionStyle](#)  
[Rows](#)

## S

[SavedBookmark](#)  
[Scrollbars](#)  
[SelectByCell](#)  
[Selected](#)  
[SelectTypeCol](#)  
[SelectTypeRow](#)  
[ShowAddButton](#)  
[ShowBookmarkButtons](#)  
[ShowDeleteButton](#)  
[ShowFindButtons](#)  
[ShowFirstLastButtons](#)  
[ShowPageButtons](#)  
[ShowPrevNextButtons](#)

ShowUpdateButton

Size

SplitterPos

SplitterVisible

Strikethrough

String

Style

StyleSet

## T

TabNavigation

TagVariant

Text

TextFormat

## U

Underline

UseDefaults

UseExactRowCount

## V

Value (Bookmark Object)

Value (Other Objects)

VertScrollBar

VisibleCols

VisibleGrps

VisibleRows

## W

WidthGap

WordWrap



## Property Pages

Sheridan Software custom controls support a feature known as property pages. Property pages provide an interface through which you can view and modify the properties of your custom control objects. The purpose of property pages is twofold. First, property pages allow you to set properties at design time that would not otherwise be available - the so-called "runtime" properties. Second, property pages allow you to modify your control in a host environment that does not provide a property sheet.

## The Property Pages Interface

The property pages provide access to a different aspect of a control's behavior. What appears in a given dialog will depend on the features that the control supports. There will always be at least one tab called 'Properties' which lists all the properties of the control. Other tabs may support added functions or utilities.

Properties are listed in a hierarchical menu structure similar to the tree view of the Windows File Manager. This structure makes it easy for you to access the properties of sub-objects and collections. As you choose a property name from the tree on the left, the valid settings for that property appear on the right, enabling you to examine or modify them.

## Accessing Property Pages

The method you use to access the property pages of your control depends on two things; the version of the control you are using, and the host environment in which you are using the control.

Many host environments support the use of the right mouse button to pop up a context-specific menu. In these environments, you simply click on your control with the right mouse button, and choose 'Property Pages' or 'Properties' from the pop-up menu.

If this behavior is not supported, use the property sheet of your design environment. You will see a property labeled '(Custom)' in the property sheet. By double-clicking this property or choosing the ellipsis (...) button, you can invoke the property pages for the selected control.

If neither of these methods are supported, you will need to consult the documentation of your host environment for information on how to change the properties of objects. You may need to choose a special menu option, or perform a shifted mouse-click or double-click on the control. Try searching your environment's online help file for references to *objects*, *embedded objects*, *object properties*, *object settings*, *OLE linking*, *OLE servers*, or *properties*.

**Note** For the SSDBCombo, SSDBDropDown, and SSDBGrid controls, property pages are replaced by the Grid Editor which performs all functions of a property page.



## ReadType Property

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Determines what data is needed by the RowBuffer object in the **UnboundReadData** event.

### Syntax

*object* . **ReadType**[ = *number* ]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying which type of unbound read to perform, as described in Settings.

### Settings

Setting	Description
0	Read all (data and bookmarks)
1	Read bookmarks only
2	Read all bookmarks and data for bound column only
3	Read all bookmarks and data for displayed column only.

There are [constants](#) available for the settings of this property.

### Remarks

This property can be used to optimize performance when using the controls in unbound mode.

This property is used during the **UnboundReadData** event to determine how the control will retrieve the unbound data. When set to 0, the control is requesting all unbound data, including bookmarks and field values. A setting of 1 means the control needs only bookmarks, such as when searching for a bookmark. A setting of 2 means the control must retrieve all bookmarks but only the data for bound column. This occurs when the control is searching for a bookmark using data from the bound column, and is limited to the DataDropDown and DataCombo controls. The last setting is for use with a DataCombo or DataDropDown that has a value set for the **DataFieldToDisplay** property. This setting will return bookmarks and the data from the field being displayed by the control.

This property is unavailable at design time and is read-only at run time.

## **ReadType Property Applies To**

ssRowBuffer

## **ReadType Property See Also**

[DataFieldToDisplay](#) property

[UnboundReadData](#) event

[Performance Tuning](#)

## ReBind Method

[Applies To](#)

[Example](#)

### Description

In bound mode, rebinds the grid to the database.

In unbound mode, refreshes the unbound grid by setting rowcount to 0 and rereading data from the top.

### Syntax

*object* . **ReBind**

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.

## **ReBind Method Applies To**

SSDBCombo

SSBBDropDown

SSDBGrid

## Record Selectors

## RecordSelectors Property

Applies To

### Description

Determines whether record selectors will be displayed.

### Syntax

*object* . **RecordSelectors**[= *boolean* ]

Part	Description
------	-------------

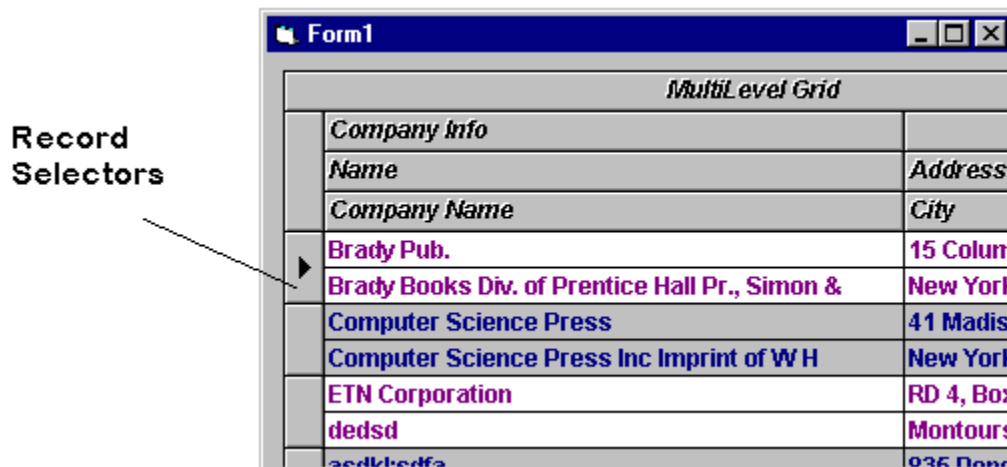
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether record selectors are displayed, as described in Settings.

### Settings

Setting	Description
---------	-------------

<b>True</b>	(Default) Record selectors will be displayed.
<b>False</b>	Record selectors will not be displayed.

### Remarks



If **RecordSelectors** is set to False, the only way the user can select an entire row is if **AllowUpdates** is set to False and **SelectByCell** is set to True.

## **RecordSelectors Property Applies To**

SSDBGrid



## Redraw Property

Applies To

Example

### Description

Determines whether the control should be redrawn during updating.

### Syntax

*object* . **Redraw**[= *boolean* ]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the control should be redrawn, as described in Settings.

### Settings

Setting	Description
<b>True</b>	(Default) The control is redrawn.
<b>False</b>	The control is not redrawn.

### Remarks

When **Redraw** is set to false, the contents of the control will not be updated after each change and will not be repainted until the **Redraw** property is set true. This is especially useful for situations where you want to perform a number of changes to the control (such as adding items to an AddItem grid) and then repainting it once to reflect these changes.

**Note** When the control is operating in AddItem mode, rows added while **Redraw** is set to False are not available until **Redraw** is reset to True, *even if accessed through code*. This is a function of the way the **Redraw** property affects data cacheing to improve performance during block additions. When **Redraw** is set to True, any cached data becomes available in the grid.

## **Redraw Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## Remove Method

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Used to remove a specified object from a collection.

### Syntax

*object* . **Remove**(*Index* **As Variant**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Index</i>	Required. An integer expression which determines the index of the object to be removed.

### Remarks

The **Remove** method is used to remove individual objects. To delete all objects in a collection, use the **RemoveAll** method.

When the **Remove** method is used, all objects that come after the deleted object will be shifted up.

## **Remove Method Applies To**

Bookmarks collection

Buttons collection

Columns collection

Groups collection

SelBookmarks collection

StyleSets collection

## **Remove Method See Also**

[Add](#) method

[Count](#) property

[RemoveAll](#) method

## RemoveAll Method (AddItem Mode)

Applies To

Example

### Description

Removes all rows from an AddItem grid.

### Syntax

*object* . **RemoveAll**

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.

## **RemoveAll Method (AddItem Mode) Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## RemoveAll Method (Collections)

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Removes all objects from a collection.

### Syntax

*object* . **RemoveAll**

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.

### Remarks

For Bookmarks, the **RemoveAll** method removes all bookmarks from a collection, setting the **Count** property to 0.

For Buttons, the **RemoveAll** method removes all buttons from a collection, setting the **IndexSelected** value to -1.

For Columns and Groups, the **RemoveAll** method removes all columns from a collection, setting the **Count** property to 0.



## **RemoveAll Method (Collections) Applies To**

Bookmarks collection

Buttons collection

Columns collection

Groups collection

SelBookmarks collection

StyleSets collection

## **RemoveAll Method (Collections) See Also**

[Add](#) method

[Remove](#) method

## RemoveAll Method (Column Object)

Applies To

Example

### Description

Removes all items from a column's combo box.

### Syntax

*object* . **RemoveAll**

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.

## **RemoveAll Method (Column Object) Applies To**

Column object

## RemoveItem Method (AddItem Mode)

[Applies To](#)

[Example](#)

### Description

Removes a string at the specified row from an AddItem grid.

### Syntax

*object* . **RemoveItem**( [*Row* **As Variant**])

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Row</i>	A variant specifying the row to remove.

### Remarks

This method is useful for removing rows one at a time. Use the **RemoveAll** method to erase an AddItem grid.

## **RemoveItem Method (AddItem Mode) Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## RemoveItem Method (Column Object)

Applies To

Example

### Description

Removes an item from a combo box.

### Syntax

*object* . **RemoveItem**(*Index* **As Integer**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>index</i>	An integer specifying the index number of the item to remove.

## **RemoveItem Method (Column Object) Applies To**

**Column** object



## Reset Method

Applies To

### Description

Destroys the associated layout for a control.

### Syntax

*object* . **Reset**

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.

### Remarks

The reset method is useful for when the programmer changes the DataSource and needs to create a new layout.

### Example

The following example resets the layout, changes the data mode, and creates a new layout:

```
SSDBGrid1.Reset  
SSDBGrid1.DataMode = 2  
SSDBGrid1.Cols = 2  
SSDBGrid1.Columns(0).Caption = "Name"  
SSDBGrid1.Columns(1).Caption = "Social Security Number"  
SSDBGrid1.Refresh           ' Needed to display new layout
```

## **Reset Method Applies To**

SSDBCombo

SSBBDropDown

SSDBGrid

## ResizeHeight Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the height of rows after the user resizes a row.

### Syntax

*object* . **ResizeHeight**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression that evaluates to the new height.

### Remarks

This property is only valid when used in the **RowResize** event procedure. This property allows you to limit the resizing of rows.

## **ResizeHeight Property Applies To**

SSDBGrid

**ResizeHeight** **See Also**

[ResizeWidth](#)

## ResizeWidth Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the width of a group or column after the user resizes it.

### Syntax

*object* . **ResizeWidth**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression that evaluates to the new width of the column or group.

### Remarks

This property is only valid when used in the **ColResize** or **GrpResize** event procedures. This property allows you to limit the resizing of columns or groups.

## **ResizeWidth Property Applies To**

SSDBGrid

## **ResizeWidth Property See Also**

[ResizeHeight](#)



## RotateText Property

[See Also](#)

[Applies To](#)

### Description

Determines whether caption text should be rotated.

### Syntax

*object* . **RotateText**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether caption area text should be rotated, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	Caption area text will be rotated.
<b>False</b>	(Default) Caption area text will not be rotated.

### Remarks

This property is most useful if the **Orientation** property is set so that the control displays vertically. If the caption is rotated and the font selected is a TrueType font, each letter of the caption is rotated 90 degrees.

## **RotateText Property Applies To**

SSDBData

## **RotateText Property See Also**

[Caption](#)

[Orientation](#)

## RoundedCorners Property

Applies To

### Description

Determines whether the control should be displayed with rounded corners or square corners.

### Syntax

*object* . **RoundedCorners**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying whether to display rounded corners, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>False</b>	Control will be displayed with square corners.
<b>True</b>	Control will be displayed with rounded corners.

### Remarks

The following is an example of a Data Command button without rounded corners:



The following is an example of a Data Command button with rounded corners:



## **RoundedCorners Property Applies To**

SSDBCommand

SSDBData

## Row Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the current display row.

### Syntax

*object* . **Row**[= *rownumber*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>rownumber</i>	An integer expression specifying the current row.

### Remarks

This property is only available at runtime.

The Row property reflects the value of the visible row, not the absolute row. See [What's New](#) for more information on the difference between visible and absolute row numbers.

To bring record into view, you have to set current bookmark. The Row property will only affect a row currently in display.

This value is zero based.

## **Row Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## Row Property See Also

Col

Grp

VisibleRows



## RowBookmark Method

[Applies To](#)

[Example](#)

### Description

Returns a bookmark of a row in the grid's display area.

### Syntax

*object* . **RowBookmark**(*RowNumb* **As Long**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>RowNumb</i>	A long integer specifying the row of the bookmark to return.

### Remarks

Bookmarks returned by this method should not be saved since the values change as visible rows change.

The bookmark returned is a variant data type.

## **RowBookmark Method Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## RowChanged Property

Applies To

### Description

Determines if any data in row has been changed. Setting to **False** performs an undo of any changes and takes the cell out of edit mode.

### Syntax

*object* . **RowChanged**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether data in the row has changed, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	Data in row has changed.
<b>False</b>	Data in row has not changed.

### Remarks

When a new row is read into the grid, this property is automatically set to false.

## **RowChanged Property Applies To**

SSDBGrid

## RowColChange Event

[See Also](#)

[Applies To](#)

### Description

Occurs just after the user changes the current row or column.

### Syntax

```
Sub control_RowColChange ([LastRow ] As Variant) [LastCol ] As Integer)
```

### The event parameters are:

Parameter	Description
<i>LastRow</i>	A variant identifying the previous row before the change. If the row has not changed, this will equal the current row.
<i>LastCol</i>	An integer identifying the previous column before the change. If the column has not changed, this will equal the current column.

### Remarks

This event is triggered just after the current column or row changes. If both the column and row change, the event will only be fired once. This event will only occur if the change was not cancelled by the **BeforeRowColChange** event

You can use this event to set up the newly entered cell. For example, if the cell is a drop-down type, you could set the **DroppedDown** property to True to automatically drop down the list when the cell is entered.

## **RowColChange Event Applies To**

SSDBGrid

## **RowColChange Event See Also**

[BeforeRowColChange](#)

[RowLoaded](#)

## RowContaining Method

[See Also](#)

[Applies To](#)

### Description

Returns the index of the row under a y-coordinate.

### Syntax

*object* . **RowContaining**(*Y* **As Single**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Y</i>	A floating point variable specifying the index of the row.

### Remarks

The value range is 0 to **VisibleRows**-1.

If the specified coordinate is out of range, an error occurs.



## **RowContaining Method Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

**RowContaining Method See Also**

[ColContaining](#)

## RowCount Property

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Returns the total number of rows requested by the object.

### Syntax

```
object . RowCount [= integer ]
```

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>integer</i>	An integer expression specifying the number of rows the Row Buffer will contain. Must be a value from 0 to 10.

### Remarks

The **RowBuffer** object is used to transfer data to a control when it is operating in unbound mode. The size of the RowBuffer is fixed at ten rows - setting the **RowCount** property to a higher value will not increase the number of rows in the RowBuffer; it will cause an error.

The main purpose of the **RowCount** property is to signal the control that all the available unbound data has been read. Generally, you will set the value of **RowCount** to zero in the **UnboundReadData** event when you begin reading data into the buffer, incrementing it with each row that is read. If there are more than ten rows of data available, the buffer will fill up, the value of **RowCount** will be set to 10 and the control will request another ten rows via the **UnboundReadData** event. Your code would reset the count of rows read to zero, and begin incrementing it again. This will continue until you reach the end of the data set.

When the last rows of data are reached, at some point the RowBuffer will contain less than ten rows of data, and your code will set the **RowCount** property to a value less than ten. This signals the control that all of the data has been read, and it will not fire another **UnboundReadData** event.

## **RowCount Property Applies To**

ssRowBuffer object

## **RowCount Property See Also**

[UnboundReadData](#) event

## RowHeight Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the height of the rows.

### Syntax

*object* . **RowHeight**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the height of every row.

### Remarks

All rows are the same height.

## **RowHeight Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

**RowHeight Property See Also**

[DefColWidth](#)



## RowLoaded Event

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Occurs when the grid loads a row of data that has scrolled into view.

### Syntax

**Sub** control\_ **RowLoaded** (*Bookmark* **As Variant**)

### The event parameters are:

Parameter	Description
<i>Bookmark</i>	A variant specifying the bookmark of the row that has been loaded.

### Remarks

If the control is used in bound mode, this event is triggered after the control has loaded the row values in memory. The event then gives you the chance to change the values of the columns.

For unbound mode, this event is triggered so that you can set the values of the text in each column.

Certain properties are affected by the **RowLoaded** event, in that where they normally affect an active row, they will affect the loaded row within the **RowLoaded** event. An example of such a property is **Text**. Any properties that behave different within the **RowLoaded** event are noted in the Remarks section of their description.

## **RowLoaded Event Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **RowLoaded Event See Also**

[RowColChange](#)

## RowNavigation Property

[See Also](#)

[Applies To](#)

### Description

Determines how the arrow keys respond when navigating rows in the grid.

### Syntax

*object* . **RowNavigation**[= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying how arrow keys work when navigation rows, as described in Settings.

### Settings

Setting	Description
0	(Default) Full navigation allowed.
1	Left and right arrow keys locked by row.
2	Up and down arrow keys locked by row.
3	All arrow keys locked by row.

There are [constants](#) available for the settings of this property.

### Remarks

Pressing the Left arrow key when in the first cell of a row moves you to the last cell of the previous row. Likewise, pressing the right arrow key when in the last cell of a row moves you to the first cell of the next row.

Pressing the Up arrow key moves you to the same cell of the previous row, pressing the Down arrow key moves you to the same cell of the next row.

## **RowNavigation Property Applies To**

SSDBGrid

**RowNavigation Property See Also**

[CellNavigation](#)

## RowOffset Property

[See Also](#)

[Applies To](#)

### Description

Determines the vertical offset used to draw the button.

### Syntax

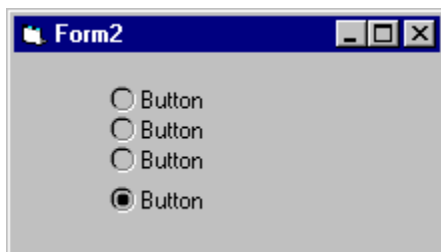
*object* . **RowOffset**[= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the horizontal offset used to draw a button.

### Remarks

Offset is determined from the button's original position. The value range for this property is -32767 to 32767 with a default value of 0.

The following example demonstrates the effect of the RowOffset property. The last button has a RowOffset value of 5:



## **RowOffset Property Applies To**

Button object

SSDBOptSet



**RowOffset Property See Also**

[ColOffset](#)

## RowResize Event

Applies To

### Description

Occurs when the user has resized the rows.

### Syntax

**Sub** control\_**RowResize** (*Cancel* **As Integer**)

**The event parameters are:**

#### **Parameter Description**

---

*Cancel*            An integer expression that specifies whether the operation occurs.

### Remarks

Setting *Cancel* = True cancels the resizing process and stops the screen from being redrawn so that it appears that the resize never occurs.

## **RowResize Event Applies To**

SSDBGrid

## RowSelectionMode Property

[See Also](#)

[Applies To](#)

### Description

Determines how a row will appear when selected.

### Syntax

*object* . **RowSelectionMode**[= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying how a row appears when selected, as described in Settings.

### Settings

Setting	Description
0	ListBox style (using System colors for highlight).
1	(Default) Invert colors.
2	3D appearance.

There are [constants](#) available for the settings of this property.

## **RowSelectionMode Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **RowSelectionMode Property See Also**

[ActiveRowStyleSet](#)

## RowTop Method

Applies To

### Description

Returns the y-coordinate of the top of a row.

### Syntax

*object* . **RowTop**(*RowNum* **As Integer**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>rownum</i>	An integer specifying the row number.

## **RowTop Method Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid



## Rows Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the number of rows in the grid.

### Syntax

*object* . **Rows**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the number of rows.

### Remarks

At runtime, **Rows** is read-only.

At design time, this property determines the amount of rows to display in Unbound or AddItem modes. When working in bound mode, this property is automatically set, deriving the information from the database.

## **Rows Property Applies To**

SSDBCombo

SSBBDropDown

SSDBGrid

## **Rows Property See Also**

[Col](#)

[Cols](#)

[Row](#)

[UnboundReadData](#)

This control is located in:

**SSDATA16.OCX, SSDATA32.OCX**

This control is located in:

**SSDATB16.OCX, SSDATB32.OCX**

## SavedBookmark Property

[Applies To](#)

[Example](#)

### Description

Sets or returns the currently saved bookmark.

### Syntax

*object* . **SavedBookmark**[= *value*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Value</i>	A variant specifying the bookmark value.

## **SavedBookmark Property Applies To**

SSDBCommand

## Scroll Event

[See Also](#)

[Applies To](#)

### Description

Occurs just before a scroll takes place.

### Syntax

**Sub** control\_ **Scroll** (*Cancel* **As Integer**)

### The event parameters are:

<b>Parameter</b>	<b>Description</b>
------------------	--------------------

---

<i>Cancel</i>	An integer expression that specifies whether the operation occurs.
---------------	--

### Remarks

A **Scroll** event can occur either by the user scrolling through the grid or through use of the **Scroll** method by the programmer.

You can prevent the scroll from occurring by setting the *Cancel* parameter to **True**. If this event is not cancelled, the scroll occurs and the **ScrollAfter** event is fired.



## **Scroll Event Applies To**

SSDBCombo

SSBBDropDown

SSDBGrid

## **Scroll Event See Also**

[Scrollbars](#)

[Scroll](#) method

[ScrollAfter](#) event

## Scroll Method

[See Also](#)

[Applies To](#)

### Description

Causes the grid to be scrolled.

### Syntax

*object* . **Scroll** *Cols* **As Integer**, *Rows* **As Long**

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Cols</i>	An integer expression specifying how many columns to scroll.
<i>Rows</i>	A long integer expression specifying how many rows to scroll.

## **Scroll Method Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **Scroll Method See Also**

[Scrollbars](#)

[Scroll event](#)

## ScrollAfter Event

[See Also](#)

[Applies To](#)

### Description

Occurs after a scroll takes place.

### Syntax

**Sub** control\_ **ScrollAfter** ()

### Remarks

You can use this event to synchronize grid scrolling with another grid or with other controls.

This event is triggered upon the successful completion of the **Scroll** event. You can prevent this event from occurring by setting the Cancel parameter of the **Scroll** event to **True**.

## **ScrollAfter Event Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **ScrollAfter Event See Also**

[Scrollbars](#)

[Scroll](#) method

[ScrollAfter](#) event



## Scrollbars Property

[See Also](#)

[Applies To](#)

### Description

Determines the type of scrollbars to use.

### Syntax

*object* . **Scrollbars**[= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the type of scrollbars to use, as described in Settings.

### Settings

Setting	Description
0	None
1	Horizontal only
2	Vertical only
3	Both
4	(Default) Automatically determined by SSDBGrid

There are [constants](#) available for the settings of this property.

## **Scrollbars Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **Scrollbars Property See Also**

[Scroll](#) event

[Scroll](#) method

## SelBookmarks Collection

Applies To

Example

### Description

The SelBookmarks collection represents a set of selected bookmark objects.

### Properties

---

Count

Item

### Methods

---

Add

Remove

RemoveAll

### Remarks

Bookmarks are added to this collection whenever a user selects a row in the grid. If Multiselect is True, then each row selected will be added to the collection in the order in which the selection occurred. Order is never based on the displayed order. When a row is unselected, that row is then removed from the **SelBookmarks** collection.

You can add bookmarks to the **SelBookmarks** collection through code. It is also easy to access the rows in the **SelBookmarks** collection without moving the current row position.

**Caution** Because of the volatile nature of the **SelBookmarks** collection, it is inadvisable to store or use the ordinal position of a row within this collection. Doing so may produce unpredictable and undesirable results.

## **SelBookmarks Collection Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## SelBookmarks Method

[See Also](#)

[Applies To](#)

### Description

Returns a Bookmark object at the specified index.

### Syntax

*object* . **SelBookmarks**( [*Index* **As Variant**])

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Index</i>	A variant specifying the bookmark number.

### Remarks

When no index is specified the SelBookmarks collection object is returned.

## **SelBookmarks Method Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **SelBookmarks Method See Also**

DeleteSelected



## SelChange Event

### Applies To

Occurs when the current range changes to a different cell or range of cells.

### Syntax

**Sub** control\_SelChange (*SelType* **As Long**, *cancel* **As Integer**, *DispSelRowOverflow* **As Integer**)

The event parameters are:

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>seltype</i>	<i>Indicates the type of selection (0=Group, 1=Column, 2=Row).</i>
<i>cancel</i>	Determines whether the selection reverts to its position before the event occurred.
<i>DispSelRowOverflow</i>	Defaults to true. If set to true, an error message is displayed if the maximum number of selected rows exceeds the MaxSelectedRows value. If set to false, no error message is displayed, giving the opportunity to display your own error message.

### Remarks

Occurs when a cell other than the current is clicked as well as when a user drags to select a new range of cells.

Setting cancel to True causes the selection to revert to the cell or range active before the event occurred.

## **SelChange Event Applies To**

SSDBGrid

## SelectByCell Property

[See Also](#)

[Applies To](#)

### Description

Determines if selection of the row will occur if the user clicks on a cell.

### Syntax

*object* . **SelectByCell**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying if the entire row should be selected when a user clicks on a cell, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) The row will be selected when a user clicks on a cell in the row.
<b>False</b>	The row will receive focus, but will not be selected (highlighted) when the user clicks on a cell.

### Remarks

This property only works when **AllowUpdate** = **False**. The behavior of this property is different from its behavior in Data Widgets 1.0.

## **SelectByCell Property Applies To**

SSDBGrid

## **SelectByCell Property See Also**

[SelectTypeCol](#)

[SelectTypeRow](#)

## SelectTypeCol Property

[See Also](#)

[Applies To](#)

### Description

Determines the column selection type.

### Syntax

*object* . **SelectTypeCol**[ = *number* ]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the column selection type, as described in Settings.

### Settings

Setting	Description
0	None
1	(Default) Single Select
2	Multi Select, Individual selection only
3	Multi-Select, Range selection allowed

There are [constants](#) available for the settings of this property.

### Remarks

Single select means that only one column can be selected at a time. Multi select means that multiple columns can be selected at once. None means that no columns can be selected.

Multiple selections may be contiguous by holding down the *Shift* key, or may be selective by holding down the *Ctrl* key when selecting. For contiguous selection, **SelectTypeCol** must be set to 3. For selective, **SelectTypeCol** must be set to either 2 or 3.

If Range Selection is allowed, code should be placed in the [SelChange](#) event to prevent extremely large selections.

## **SelectTypeCol Property Applies To**

SSDBGrid

**SelectTypeCol Property See Also**

[SelectTypeRow](#)



## SelectTypeRow Property

[See Also](#)

[Applies To](#)

### Description

Determines the row selection type.

### Syntax

*object* . **SelectTypeRow**[= *number* ]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the row selection type, as described in Settings.

Settings	Setting	Description
0	None	
1	Single Select	
2	(Default) Multi Select, Individual selection only	
3	Multi-Select, Range selection allowed	

There are [constants](#) available for the settings of this property.

### Remarks

None means that no rows can be selected. Single select means that only one row can be selected at a time. Multi select means that multiple row can be selected as once. The number of rows that may be selected is determined by the [MaxSelectedRows](#) property.

Multiple selections may be contiguous by holding down the *Shift* key, or may be selective by holding down the *Ctrl* key when selecting. For contiguous selection, **SelectTypeRow** must be set to 3. For selective, **SelectTypeRow** must be set to either 2 or 3.

If Range Selection is allowed, code should be placed in the [SelChange](#) event to prevent extremely large selections.

## **SelectTypeRow Property Applies To**

SSDBGrid

## **SelectTypeRow Property See Also**

SelectTypeCol property

MaxSelectedRows property

## Selected Property

Applies To

### Description

Sets or returns whether an object is selected.

### Syntax

*object* . **Selected**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the object is selected, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	Object is currently selected.
<b>False</b>	Object is not currently selected.

## **Selected Property Applies To**

Column object

Group object

## ShowAddButton Property

[See Also](#)

[Applies To](#)

### Description

Determines whether the Add Record button is displayed on the control.

### Syntax

*object* . **ShowAddButton**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying the display state of the Add Record button on the control, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) The Add Record button will be displayed.
<b>False</b>	The Add Record button will not be displayed.

## **ShowAddButton Property Applies To**

SSDBData

## **ShowAddButton Property See Also**

[ShowBookmarkButtons](#)

[ShowCancelButton](#)

[ShowDeleteButton](#)

[ShowFindButtons](#)

[ShowFirstLastButtons](#)

[ShowPageButtons](#)

[ShowPrevNextButtons](#)

[ShowUpdateButton](#)



## ShowBookmarkButtons Property

[See Also](#)

[Applies To](#)

### Description

Determines which of the bookmark buttons are to be displayed on the control.

### Syntax

*object* . **ShowBookmarkButtons**[= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying which of the bookmark buttons are to be displayed, as described in Settings.

### Settings

Setting	Description
0	Show none of the bookmark buttons
1	Show Add bookmark button only
2	Show Goto bookmark button only
3	Show Add and Goto bookmark buttons only
4	Show Clear All Bookmarks button only
5	Show Add and Clear All Bookmarks buttons only
6	Show Clear All and Goto Bookmark buttons only
7	(Default) Show all bookmark buttons

There are [constants](#) available for the settings of this property.

## **ShowBookmarkButtons Property Applies To**

SSDBData

## **ShowBookmarkButtons Property See Also**

[ShowAddButton](#)

[ShowCancelButton](#)

[ShowDeleteButton](#)

[ShowFindButtons](#)

[ShowFirstLastButtons](#)

[ShowPageButtons](#)

[ShowPrevNextButtons](#)

[ShowUpdateButton](#)

## ShowBookmarkDropDown Event

[See Also](#)

[Applies To](#)

### Description

Occurs immediately before the dropdown bookmark list is displayed.

### Syntax

**Sub** control\_ **ShowBookmarkDropDown**()

### Remarks

If the property **DroppedDown** is set to TRUE, the SSDBData control will drop down the list. If **DroppedDown** is set to FALSE, the dropdown will not be displayed.

## **ShowBookmarkDropDown Event Applies To**

SSDBData

## **ShowBookmarkDropDown Event See Also**

[DroppedDown](#)

[CloseBookmarkDropDown](#) Method

## ShowCancelButton

[See Also](#)

[Applies To](#)

### Description

Determines whether the Cancel button is displayed on the control.

### Syntax

*object* . **ShowCancelButton**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying the display state of the Cancel button on the control.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) The Cancel button will be displayed.
<b>False</b>	The Cancel button will not be displayed.

### Remarks

The Cancel (Cancel Add) button cancels the adding of a new record to the database.

## **ShowCancelButton Applies To**

SSDBData



## **ShowCancelButton See Also**

[ShowAddButton](#)

[ShowBookmarkButtons](#)

[ShowDeleteButton](#)

[ShowFindButtons](#)

[ShowFirstLastButtons](#)

[ShowPageButtons](#)

[ShowPrevNextButtons](#)

[ShowUpdateButton](#)

## ShowDeleteButton Property

[See Also](#)

[Applies To](#)

### Description

Determines whether the Delete Record button is displayed on the control.

### Syntax

*object* . **ShowDeleteButton**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying the display state of the Delete Record button on the control, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) The Delete Record button will be displayed.
<b>False</b>	The Delete Record button will not be displayed.

## **ShowDeleteButton Property Applies To**

SSDBData

## **ShowDeleteButton Property See Also**

[ShowAddButton](#)

[ShowBookmarkButtons](#)

[ShowCancelButton](#)

[ShowFindButtons](#)

[ShowFirstLastButtons](#)

[ShowPageButtons](#)

[ShowPrevNextButtons](#)

[ShowUpdateButton](#)

## ShowFindButtons Property

[See Also](#)

[Applies To](#)

### Description

Determines which of the find buttons are to be displayed on the control.

### Syntax

*object* . **ShowFindButtons**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying which of the find buttons are to be displayed, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
0	Show none of the find buttons
1	Show Find button only
2	Show Find and Find Next buttons only
3	Show Find, Previous, and Next buttons

There are [constants](#) available for the settings of this property.

## **ShowFindButtons Property Applies To**

SSDBData

## **ShowFindButtons Property See Also**

[ShowAddButton](#)

[ShowBookmarkButtons](#)

[ShowCancelButton](#)

[ShowDeleteButton](#)

[ShowFirstLastButtons](#)

[ShowPageButtons](#)

[ShowPrevNextButtons](#)

[ShowUpdateButton](#)

## ShowFindDialog Event

[See Also](#)

[Applies To](#)

### Description

Occurs when the Find dialog is called, immediately prior to displaying the dialog.

### Syntax

**Sub** control\_ **ShowFindDialog**()



## **ShowFindDialog Event Applies To**

SSDBData

## **ShowFindDialog Event See Also**

[FindDialog](#)

[CloseFindDialog](#) event

## ShowFirstLastButtons Property

[See Also](#)

[Applies To](#)

### Description

Determines whether the first and last record button is displayed on the control.

### Syntax

*object* . **ShowFirstLastButtons**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying the display state of the First Record and Last Record buttons on the control, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) The First Record and Last Record buttons will be displayed.
<b>False</b>	The First Record and Last Record buttons will not be displayed.

### Remarks

Clicking the First Record button will take the user to the first record in the database, while clicking the Last Record button will take the user to the last record in the database.

## **ShowFirstLastButtons Property Applies To**

SSDBData

## **ShowFirstLastButtons Property See Also**

[ShowAddButton](#)

[ShowBookmarkButtons](#)

[ShowCancelButton](#)

[ShowDeleteButton](#)

[ShowFindButtons](#)

[ShowPageButtons](#)

[ShowPrevNextButtons](#)

[ShowUpdateButton](#)

## ShowPageButtons Property

[See Also](#)

[Applies To](#)

### Description

Determines whether the page forward and page backward buttons are displayed on the control.

### Syntax

*object* . **ShowPageButtons**[ = *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying the display state of the Page forward and Page backward buttons on the control, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) The Page forward and Page backward buttons will be displayed.
<b>False</b>	The Page forward and Page backward buttons will not be displayed.

### Remarks

Clicking the Page Forward button will move *n* records forward through the data. Clicking the Page Backward button will move *n* records backward through the data. The value for *n* is determined by the [PageValue](#) property.

## **ShowPageButtons Property Applies To**

SSDBData

## **ShowPageButtons Property See Also**

[ShowAddButton](#)

[ShowBookmarkButtons](#)

[ShowCancelButton](#)

[ShowDeleteButton](#)

[ShowFindButtons](#)

[ShowFirstLastButtons](#)

[ShowPrevNextButtons](#)

[ShowUpdateButton](#)



## ShowPrevNextButtons Property

[See Also](#)

[Applies To](#)

### Description

Determines whether the previous record and next record buttons are displayed on the control.

### Syntax

*object* . **ShowPrevNextButtons**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying the display state of the Previous Record and Next Record buttons on the control, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) The Previous Record and Next Record buttons will be displayed.
<b>False</b>	The Previous Record and Next Record buttons will not be displayed.

## **ShowPrevNextButtons Property Applies To**

SSDBData

## **ShowPrevNextButtons Property See Also**

[ShowAddButton](#)

[ShowBookmarkButtons](#)

[ShowCancelButton](#)

[ShowDeleteButton](#)

[ShowFindButtons](#)

[ShowFirstLastButtons](#)

[ShowPageButtons](#)

[ShowUpdateButton](#)

## ShowUpdateButton Property

[See Also](#)

[Applies To](#)

### Description

Determines whether the Update Record button is displayed on the control.

### Syntax

*object* . **ShowUpdateButton**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying the display state of the Update Record button on the control, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) The Update Record will be displayed.
<b>False</b>	The Update Record will not be displayed.

### Remarks

The Update Record button will write any changes made to the most recently modified record.

## **ShowUpdateButton Property Applies To**

SSDBData

## **ShowUpdateButton Property See Also**

[ShowAddButton](#)

[ShowBookmarkButtons](#)

[ShowCancelButton](#)

[ShowDeleteButton](#)

[ShowFindButtons](#)

[ShowFirstLastButtons](#)

[ShowPageButtons](#)

[ShowPrevNextButtons](#)

## Size Property

Applies To

Example

### Description

Returns or sets the font size used in the specified **Font** or **Headfont** object.

### Syntax

*object* . **Size**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Font</i>	An integer expression specifying the size of the font in points.

### Remarks

Use this property to format text in the font size you want. The default font size is determined by the operating system. To change the size, specify the size of the font in points. The maximum value for the **Size** property is 2048 points.

The **Font** and **Headfont** objects are not directly available at design time. At design time, set the **Size** property through the control's **Font** or **Headfont** property. At runtime, you can set **Size** directly by specifying its setting for the appropriate **Font/Headfont** object.

## **Size Property Applies To**

Font object

HeadFont object



## **Soundex Method Applies To**

SSDBData

## Soundex Method

Applies To

### Description

Returns the soundex string for a supplied string.

### Syntax

*object* . **Soundex**(*String* **As String**)

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>String</i>	A string expression for which you wish to find the Soundex string.

### Remarks

A Soundex string is a 4-character code that represents the supplied string. This code can be used to search for words that are phonetically similar.

Soundex codes are based on an algorithm that analyzes the letters in a string and returns a value. Words with similar values will have similar sounds. Even though it considers phonetics, the Soundex algorithm is ultimately based on spelling, so words with the same pronunciation may return different Soundex codes. For example, the word "THROUGH" returns a Soundex code of T620. The word "THREW" returns a Soundex code of T600.

Through code, you can compare Soundex values to see if they are within a certain range, and take action to either exclude or include particular strings. Soundex is particularly useful when searching for names. For example, "Smith", "Smyth" and "Smythe" all return Soundex codes of S530. "Cook" and "Koch" both return identical Soundex values, even though they are spelled quite differently.

## Speed Buttons

Speed buttons allow for a user to click on a button and hold it to repeat a function.

This function is controlled by the **DelayInitial** and **DelaySubsequent** properties.

**DelayInitial** determines the amount of time before a speed button begins to repeat when the mouse button is held down. **DelaySubsequent** determines the amount of time between subsequent clicks are repeated when the mouse button is held down on a repeatable button.

## SplitterMove Event

[See Also](#)

[Applies To](#)

### Description

Occurs when the splitter is relocated by the user.

### Syntax

**Sub** control\_ **SplitterMove** (*Cancel* **As Integer**)

### The event parameters are:

<b>Parameter</b>	<b>Description</b>
------------------	--------------------

---

<i>Cancel</i>	An integer expression that specifies whether the operation occurs.
---------------	--

### Remarks

Setting *Cancel* = 1 causes the splitter to return to its previous position and the screen is not redrawn so that it appears that the process never happened.

## **SplitterMove Event Applies To**

SSDBGrid

## **SplitterMove Event See Also**

[SplitterPos](#)

[SplitterVisible](#)

## SplitterPos Property

[See Also](#)

[Applies To](#)

### Description

Sets or returns the column in which to place the splitter.

### Syntax

*object* . **SplitterPos**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the position of the splitter.

### Remarks

Valid range is from 0 to the maximum number of columns created.

## **SplitterPos Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid



## **SplitterPos Property See Also**

[SplitterVisible](#)

[SplitterMove](#) Event

## SplitterVisible Property

[See Also](#)

[Applies To](#)

### Description

Determines whether the splitter is visible.

### Syntax

*object* . **SplitterVisible**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying the display state of the splitter, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	The splitter will be displayed.
<b>False</b>	(Default) The splitter will not be displayed.

### Remarks

The splitter is used to fix the position of groups and columns. When the splitter is visible, the user can relocate it by using the mouse.

## **SplitterVisible Property Applies To**

SSDBGrid

## **SplitterVisible Property See Also**

[SplitterPos](#)

[SplitterMove](#) Event

**Standard Method** - Depending on your host environment, this method may be referred to by a different name or may not apply to this control. Refer to your host environment's documentation or help file for further information regarding this method.

**Standard Property** - Depending on your host environment, this property may be referred to by a different name or may not apply to this control. Refer to your host environment's documentation or help file for further information regarding this property.

## Strikethrough Property

Applies To

Example

### Description

Returns or sets the font style of the specified **Font** or **Headfont** object to either strikethrough or non-strikethrough.

### Syntax

*object* . **Strikethrough**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Font</i>	A Boolean expression specifying the font style as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	Turns on strikethrough formatting
<b>False</b>	(Default) Turns off strikethrough formatting

### Remarks

The **Font** and **Headfont** objects are not directly available at design time. At design time, set the **Strikethrough** property through the control's **Font** or **Headfont** property. At runtime, you can set **Strikethrough** directly by specifying its setting for the appropriate **Font/Headfont** object.

## **Strikethrough Property Applies To**

Font object

HeadFont Object



## String Property (Bookmark Object)

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Returns the string stored in a bookmark object.

### Syntax

*object* . **String**[= *string*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>text</i>	A string expression that evaluates to the string stored in a bookmark.

## **String Property Applies To**

Bookmark object (SSDBData only)

## **String Property See Also**

[Value](#)

## Style Property

Applies To

### Description

Sets or returns the column's control style.

### Syntax

*object* . **Style**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the control style as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
0	(Default) Edit Mode
1	Edit Button
2	Check Box
3	Combo Box
4	Button

There are constants available for the settings of this property.

### Remarks

The following are examples of the various modes that **Style** can be set to:

<i>Address Info</i>		
<i>Address</i>		
<i>City</i>	<i>State</i>	<i>Zip</i>
35 Pinelawn Road		
Meville	NY	11747
Rte 128		
Reading	MA	01867
666 Fifth Ave		
New York	NY	10103
390 Bridge Pkwy.		
Redwood City	CA	94065
15 Columbus Cir.		

Edit (**Address** field)

<i>Other Info</i>		
<i>Telephone</i>	<i>Fax</i>	
<i>Comments</i>		<i>Paid</i>
212-869-7440		<input checked="" type="checkbox"/>
617-944-3700	617-964-9460	<input type="checkbox"/>
800-223-6834	212-765-3869	
GENERAL TRADE BOOKS -		<input checked="" type="checkbox"/>
800-950-2665	415-594-4409	<input checked="" type="checkbox"/>
212-373-8093	212-373-8292	

CheckBox (**Paid** field)

Form1

*Address Info*

*Address*

City	State	Zip
35 Pinela		
Mehville	NY	11747
Alameda		
Albany	MA	01867
Atlanta		
Belmont		
Baldwinsville	NY	10103
Berkeley		
Blue Ridge Summit	CA	94065
15 Columbus Cir.		

Combo Box (**City** field)

Form1

*Company Info*

*Name*

*Company Name*

ACM
Association for Computing Machinery
Addison-Wesley
Addison-Wesley Publishing Co Inc.
Bantam Books
Bantam Books Div of: Bantam
Benjamin/Cummings
Benjamin-Cummings Publishing
Brady Pub.

Button (**Name** field)

## **Style Property Applies To**

Column object

## StyleSet Object

[See Also](#)

[Applies To](#)

### Description

The **StyleSet** object contains properties pertaining to the appearance of the **ActiveCell**, **Column**, and **Group** objects as well as the SSDBCombo, SSDBDropDown, and SSDBGrid controls.

### Properties

---

<a href="#">AlignmentPicture</a>	<a href="#">ForeColor</a>	<a href="#">PictureMetaHeight</a>
<a href="#">AlignmentText</a>	<a href="#">Name</a>	<a href="#">PictureMetaWidth</a>
<a href="#">BackColor</a>	<a href="#">Picture</a>	
<a href="#">Font</a>		

### Remarks

You identify a StyleSet by using the **[StyleSet](#)** or **[HeadStyleSet](#)** properties.



## **StyleSet Object Applies To**

StyleSets collection

## **StyleSet Object See Also**

[HeadStyleSet](#)

[StyleSet](#)

[StyleSets](#) collection.

## StyleSet Property

[See Also](#)

[Applies To](#)

### Description

Returns or sets the name of a **StyleSet** in the **StyleSets** collection.

### Syntax

*object* . **StyleSet**[= *text*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>text</i>	A string expression that evaluates to the name of a StyleSet.

### Remarks

This property determines the StyleSet to be used for the controls and objects listed in the *Applies To* list. Note that the **StyleSet** specified must be in the **StyleSets** collection. If a change is made to a StyleSet, the control must be refreshed. StyleSets will override each other based on the following hierarchy:

#### **Data Area:**

<b>ActiveCell.StyleSet</b>	(overrides all below)
<b>Control.ActiveRowStyleSet</b>	(overrides all below)
<b>Column.StyleSet</b>	(overrides all below)
<b>Group.StyleSet</b>	(overrides all below)
<b>Control.StyleSet</b>	

The following is a list of properties used in the various StyleSets.

#### **Properties Used by SSDBCombo, SSDBDropDown, SSDBGrid**

[BackColor](#)      [Font](#)      [ForeColor](#)

#### **Properties Used by the Column, Group, and ActiveCell objects**

[BackColor](#)      [Font](#)      [ForeColor](#)

## **StyleSet Property Applies To**

ActiveCell object

Column object

Group object

SSDBCombo

SSDBDropDown

SSDBGrid

## **StyleSet Property See Also**

[HeadStyleSet](#)

[StyleSet](#) object

[StyleSets](#) collection

## StyleSets Collection

[See Also](#)

[Applies To](#)

### Description

Contains a collection of **StyleSet** objects

### Properties

---

[Count](#)

[Item](#)

### Methods

---

[Add](#)

[Remove](#)

[RemoveAll](#)

### Syntax

*object* . **StyleSets**

### Remarks

The StyleSet properties can be set to distinguish one StyleSet from another. A StyleSet called "Loss" may have a **BackColor** of 'Red', while a StyleSet called "Profit" may have a **BackColor** of 'Green'.

Note that not all properties of a StyleSet are used in every case.

## **StyleSets Collection Applies To**

Column object

Group object

SSDBCombo

SSDBDropDown

SSDBGrid

## **StyleSets Collection See Also**

[HeadStyleSet](#)

[StyleSet](#)

[StyleSet](#) object



## StyleSets Method

Applies To

### Description

Returns a StyleSet object at the specified index.

### Syntax

*object* . **StyleSets**( [*Index As Variant*])

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Index</i>	A variant specifying the StyleSet number.

### Remarks

When no index is specified the StyleSets collection object is returned.

## **StyleSets Method Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## Support files needed for distribution - 16 Bit

Due to the nature of the OLE architecture, Data Widgets 2.0 controls require that the following supporting files be shipped with your application.

These files must be installed on any machine that uses Data Widgets.

OC25.DLL  
TYPELIB.DLL

In addition to being installed, the following file(s) must be registered, either by a setup program or by using the 16-bit Registration Server Utility (REGSVR.EXE) available from Microsoft.

OC25.DLL

When you install the Data Widgets package, the correct files are automatically installed and registered on your machine, provided you do not have later versions installed.

## A note about OLE file distribution

The introduction of OCX controls and the availability of 32-bit Windows platforms has introduced new concerns regarding the five files used for OLE (COMPOBJ.DLL, OLE2.DLL, OLE2DISP.DLL, OLE2NLS.DLL, STORAGE.DLL). **These files are only needed to support 16-Bit applications created with Data Widgets.**

## Windows 95 and Windows NT

If your application is running under Windows 95 or Windows NT, then the OLE DLLs are part of the OS and you do not need to install or update these files, provided the version numbers match or exceed those below:

### Windows 95

---

COMPOBJ.DLL	Version 2.2
OLE2.DLL	Version 2.2
OLE2DISP.DLL	Version 2.1
OLE2NLS.DLL	Version 2.1
STORAGE.DLL	Version 2.2

### Windows NT

---

COMPOBJ.DLL	Version 2.1
OLE2.DLL	Version 2.1
OLE2DISP.DLL	Version 2.1
OLE2NLS.DLL	Version 2.1
STORAGE.DLL	Version 2.1

## Windows 3.x and Windows for Workgroups 3.x

If your application is running under Windows 3.x or Windows For Workgroups 3.x, you **must** make sure that these DLLs are installed and registered for any applications created with the 2.0 version of Data Widgets. These DLLs are included with the Data Widgets installation disks and are copied to your machine when you install Data Widgets under either of these environments.

### Windows 3.x and Windows for Workgroups 3.x

---

COMPOBJ.DLL	Version 2.03
OLE2.DLL	Version 2.03
OLE2DISP.DLL	Version 2.03
OLE2NLS.DLL	Version 2.03
STORAGE.DLL	Version 2.03

## Support files needed for distribution - 32 Bit

Due to the nature of the OLE architecture, Data Widgets 2.0 controls require that a number of supporting files be shipped with your application. These files must be installed on any machine that runs a Data Widgets application.

MFC42.DLL	4.2.6256
MSVCRT.DLL	4.20.6201
OLEAUT32.DLL	2.20.4054
OLEPRO32.DLL	5.0.4055

The above files are automatically installed and registered on your machine by the Data Widgets package, provided you do not have later versions installed.

Additionally, the following files must be registered, either via a setup program or via the 32-bit Registration Server Utility (REGSVR32.EXE) available from Microsoft:

MFC42.DLL  
OLEAUT32.DLL  
OLEPRO32.DLL

## **System Requirements**

You must have the following to utilize Data Widgets:

- § Microsoft Visual Basic version 4.x or a development tool that supports OLE Custom Controls (.OCX files).
- § A hard disk with at least 5 megabytes of available space for a full installation.
- § For the 32-bit version of Data Widgets, you must have Windows 95 or later, or Windows NT 3.51 or later.
- § For the 16-bit version of Data Widgets, you must have Windows version 3.1 or later, running in enhanced mode.

## TabNavigation Property

Applies To

### Description

Determines how the grid will process the Tab key.

### Syntax

*object* . **TabNavigation**[= *number* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression that specifies how the control will handle Tab key usage, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
0	(Default) Move to Next Cell. The Tab key will cause the cursor to move to the next cell in the grid, or to the next line in the grid if the focus is on the last cell in a row.
1	Move to Next Control. The Tab key will move the focus to the next control on the form, as determined by the tab order.

### Remarks

This property gives you the ability to change the way the SSDBGrid control responds to the Tab key. This property is particularly useful if you are using the Grid as a multi-column list box.

There are constants available for the settings of this property.

## **TabNavigation Property Applies To**

SSDBGrid

## TagVariant Property

[Applies To](#)

[Example](#)

### Description

Stores any extra data needed for your program. You can use this property to attach data of any type, except user-defined types, to an object or control.

### Syntax

*object* . **TagVariant**[= *expression*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>text</i>	A Variant expression.

### Remarks

The **TagVariant** property is similar to the Visual Basic **Tag** property. However, in addition to string expressions, the **TagVariant** property can store any data type including other objects, with the exception of user-defined types.

This property is only available at runtime.



## **TagVariant Property Applies To**

Column object

Group object

SSDBCombo

SSDBCommand

SSDBDropDown

SSDBGrid

SSDBOptSet



## Technical Information



### **Performance Tuning**

Methods you can use to improve the speed of your applications when using Data Widgets 2.0.



### **Using Data Widgets 2.0 with Visual C++**

Information and examples to get you started using the Data Widgets controls in Microsoft Visual C++.



### **System Requirements**

A list of requirements for using the product



### **Included Files**

A list of files included with the product.



### **Distribution Notes**

A list of the **files you need to distribute** with your applications



## Technical Support

### **World Wide Web**

The Sheridan Software World Wide Web site provides the latest patches and product information, as well as information for the Visual Basic developer.

<http://www.shersoft.com>

### **Internet Email**

Submit your questions to our technical support staff via electronic mail. Be sure to include detailed information on your problem, the Sheridan product and product version you are using, as well as information on your host environment such as the machine type, RAM, video card, and operating system.

[support.data@shersoft.com](mailto:support.data@shersoft.com)

### **CompuServe**

You can obtain technical support on CompuServe by contacting the SYSOP at the SHERIDAN section of the COMPA forum. You can type **GO SHERIDAN** at any CompuServe prompt.

### **Fax**

To fax questions or comments regarding any Sheridan product, dial **(516) 753-3661**.

### **Telephone Support**

For technical support for this or any other Sheridan product, contact Sheridan Software

systems at (516) 753-0985. You can either speak to a live technical support representative or get answers using the Automated Fax Service.

## Text Property

Applies To

### Description

Sets or returns the text string associated with the specified column, or in the case of the SSDBCombo, text currently in the edit portion.

### Syntax

*object* . **Text**[= *text*]

Part	Description
------	-------------

<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
---------------	---

<i>text</i>	A string expression that evaluates to the string stored in the column.
-------------	--

### Remarks

	<i>Company Info</i>	<i>Address Info</i>		
	<i>Name</i>	<i>Address</i>		
	<i>Company Name</i>	<i>City</i>	<i>State</i>	<i>Zip</i>
▶	Microsoft Press	One Microsoft Way		
	Microsoft Press Div of: Microsoft	Redmond	WA	98052-6399
	Morgan Kaufmann	2020 Campus Dr. Suite 260		

Based on the above graphic, the following is true:

```
?SSDBGrid.Columns(0).Text = "Microsoft Press"
```

```
?SSDBGrid.Columns(3).Text = "Redmond"
```

## **Text Property Applies To**

ActiveCell object

Column object

SSDBCombo

## TextError Event

[See Also](#)

[Applies To](#)

### Description

Occurs when text fails validation.

### Syntax

```
Sub control_TextError (ErrCode As Long, ErrString As String, RestoreString As String, Text As String, RtnDispErrMsg As Integer, RtnRestore As Integer)
```

### The event parameters are:

#### Parameter Description

---

<i>ErrCode</i>	A long integer specifying the error code resulting from the validation failure.
<i>ErrString</i>	A string that evaluates to the error message that SSDBCombo generates.
<i>RestoreString</i>	A string that evaluates to the restore string if RtnRestore is true.
<i>Text</i>	A string that evaluates to the text that failed validation.
<i>RtnDispErrMsg</i>	An integer expression that indicates if an error message box should be displayed.
<i>RtnRestore</i>	An integer expression that determines if text should be restored to its original state.

## **TextError Event Applies To**

SSDBCombo

SSBBDropDown

## **TextError Event See Also**

[ListAutoValidate](#)



## TextFormat Property

[Applies To](#)

[Example](#)

### Description

Used to set the format mask of the edit portion of the control.

### Syntax

*object* . **TextFormat**[= *string*]

### Settings

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>string</i>	A string expression that determines the format mask to use.

### Remarks

This property complies with standard text formatting properties.

## **TextFormat Property Applies To**

SSDBCombo

## Totalling Values in a Grid column

This example uses a DataGrid that is bound to a recordsource. The value to be totalled is in the first column of the grid (Columns(0).) Totalling the values in the column is accomplished by clicking a command button.

The value of the cell in the column is examined for each row in the grid, and added to a temporary variable. When all rows have been added, the value of the variable is assigned to the control displaying the total (a text box.) The following code is used:

```
(General) (declarations)
Dim iTot as Global Integer

Private Sub Command1_Click()

Dim iC as Integer
Dim vBM as Variant

    SSDBGrid1.MoveFirst
    For iC = 0 To SSDBGrid1.Rows - 1
        vBM = SSDBGrid1.GetBookmark(i)
        iTot = iTot + SSDBGrid1.Columns(0).CellValue(vBM)
    Next iC

    Text1.Text = iTot

End Sub
```

This code uses the Bookmark value of each row in combination with the **CellValue** property of the Column object to examine the value of the cell.

## UnboundAddData Event

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Occurs when an unbound grid has a new row added to it.

### Syntax

**Sub** control\_ **UnboundAddData** (*RowBuf* **As** **ssRowBuffer**, *NewRowBookmark* **As** **Variant**)

### The event parameters are:

#### Parameter Description

---

*RowBuf* The **ssRowBuffer** object that will be used to transfer the data. The number of rows to be retrieved is determined by **RowCount**.

*NewRowBookmark* A bookmark that acts as a unique identifier for each row of data.

### Remarks

This event notifies you when a new row of data must be added. The *RowBuf* argument represents an **ssRowBuffer** object that can contain up to ten rows of data to be transferred between the control and the data source. Before returning from this event, *newrowbookmark* must be set to the bookmark of the newly added row.

## **UnboundAddData Event Applies To**

SSDBGrid

## **UnboundAddData Event See Also**

[AllowAddNew](#) property

[UnboundPositionData](#) event

[UnboundReadData](#) event

[UnboundWriteData](#) event

[ssRowBuffer](#) Object

## UnboundDeleteRow Event

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Occurs when a row is deleted from an unbound grid.

### Syntax

**Sub** control\_ **UnboundDeleteRow** (*Bookmark* **As Variant**)

### The event parameters are:

<b>Parameter</b>	<b>Description</b>
------------------	--------------------

---

<i>Bookmark</i>	A bookmark of the row to be deleted.
-----------------	--------------------------------------

### Remarks

This event notifies you that a row must be deleted from the database. The *Bookmark* argument must be set to the bookmark value provided when the row was retrieved with the **UnboundReadData** event or added by **UnboundAddData** event.

## **UnboundDeleteRow Event Applies To**

SSDBGrid



## **UnboundDeleteRow Event See Also**

[AllowDelete](#) property

[UnboundAddData](#) event

[UnboundPositionData](#) event

[UnboundReadData](#) event

[UnboundWriteData](#) event

[ssRowBuffer](#) Object

## UnboundPositionData Event

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Occurs when a user repositions the unbound grid.

### Syntax

**Sub** control\_ **UnboundPositionData** (*StartLocation* **As Variant**, *NumberOfRowsToMove* **As Long**, *NewLocation* **As Variant**)

### The event parameters are:

#### Parameter Description

---

*StartLocation* A bookmark specifying the row before (for forward) or after (for reverse) the rows to be retrieved. If null, forward fetching occurs from the beginning of the data set while reverse begins at the end of the data set.

*NumberOfRowsToMove* A long integer specifying the number of rows to move. A positive number indicates a move forward while a negative number indicates a move backwards.

*NewLocation* A bookmark specifying the new location to be repositioned to.

### Remarks

**UnboundPositionData** is used to optimize data display. Under normal circumstances, the grid is read sequentially when a user scrolls the grid, but with **UnboundPositionData**, records that will not be displayed can be bypassed for quicker data access. *StartLocation* and *NumberOfRowsToMove* are provided by the control, allowing you to specify *NewLocation*.

## **UnboundPositionData Event Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **UnboundPositionData Event See Also**

[UnboundAddData](#) event

[UnboundDeleteRow](#) event

[UnboundReadData](#) event

[UnboundWriteData](#) event

[ssRowBuffer](#) Object

## UnboundReadData Event

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Occurs when an unbound grid requests data for display.

### Syntax

**Sub** control\_ **UnboundReadData** (*RowBuf* **As** *ssRowBuffer*, *StartLocation* **As** *Variant*, *ReadPriorRows* **As** *Boolean*)

### The event parameters are:

#### Parameter Description

---

<i>RowBuf</i>	The <i>ssRowBuffer</i> object that will contain the retrieved data. The number of rows to be retrieved is determined by <b>RowCount</b> .
<i>StartLocation</i>	A bookmark specifying the row before (for forward) or after (for reverse) the rows to be retrieved. If null, forward fetching occurs from the beginning of the data set while reverse begins at the end of the data set.
<i>ReadPriorRows</i>	If set to <b>False</b> , data is retrieved in a forward direction, if <b>True</b> , data is retrieved in a reverse direction.

### Remarks

This event notifies you that the control must retrieve data. Once retrieved, data is stored in the control's buffer.

In unbound mode, the control uses the *ssRowBuffer* object to populate the grid. You use the **UnboundReadData** event to fill the RowBuffer with data. The RowBuffer retrieves a maximum of ten rows of data at a time. As long as the RowBuffer is filled with ten rows of data, the control will continue to fire the **UnboundReadData** event.

When all the data has been read into the control, you signal the control to stop reading by setting the **RowCount** property of the RowBuffer object to a value less than ten. The control will then stop firing the **UnboundReadData** event.

While the unbound read is occurring, the control uses the *Rows* property to determine the size of the grid and position the scroll bar. Before the unbound read begins, the control estimates the size of the data set to be 100 rows. During the unbound read, the control will adjust the scrollbar according to its estimate. The estimate changes when:

- The control reads more than 100 rows of data. The estimate is then enlarged to 200 rows. After 200 rows have been read, it changes to 300 rows, and so on.
- The **RowCount** of the *RowBuffer* object is set to less than ten. At this point the control has read all the data and knows exactly how many rows there are, and will adjust itself accordingly.

## **UnboundReadData Event Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **UnboundReadData Event See Also**

[UnboundAddData](#) event

[UnboundDeleteRow](#) event

[UnboundPositionData](#) event

[UnboundWriteData](#) event

[ssRowBuffer](#) Object

[Performance Tuning](#)

## UnboundWriteData Event

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Occurs when an unbound grid has a row of modified data to write to the database.

### Syntax

**Sub** control\_ **UnboundWriteData** (*RowBuf* **As** **ssRowBuffer**, *writelocation* **As Variant**)

### The event parameters are:

<b>Parameter</b>	<b>Description</b>
<i>RowBuf</i>	The <b>ssRowBuffer</b> object that contains the modified data.
<i>WriteLocation</i>	A value identifying the unique bookmark of the row as specified in the <b>UnboundReadData</b> and <b>UnboundWriteData</b> events.

### Remarks

During this event, the value of the **RowCount** property of the **ssRowBuffer** object will always be 1 since you can update only one row of a data at a time.

To cancel the **UnboundWriteData** event, set the **RowCount** property of the **ssRowBuffer** object to 0.



## **UnboundWriteData Event Applies To**

SSDBGrid

## **UnboundWriteData Event See Also**

[UnboundAddData](#) event

[UnboundDeleteRow](#) event

[UnboundPositionData](#) event

[UnboundReadData](#) event

[UnboundWriteData](#) event

[ssRowBuffer](#) Object

## Underline Property

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Returns or sets the font style of the specified **Font** or **HeadFont** object to either underlined or non-underlined.

### Syntax

*object* . **Underline**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying the font style as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	Turns on underline formatting.
<b>False</b>	(Default) Turns off underline formatting.

### Remarks

The **Font** and **Headfont** objects are not directly available at design time. At design time, set the **Underline** property through the control's **Font** or **HeadFont** property. At runtime, you can set **Underline** directly by specifying its setting for the appropriate **Font/Headfont** object.

## **Underline Property Applies To**

Font object

Headfont object

## **Underline Property See Also**

Font object

Headfont object

## Update Method

Applies To

### Description

Updates any modified information in the grid. This method is applicable to all data modes.

### Syntax

*object* . **Update**

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.

### Remarks

This method causes the control to update the database with any data that has been modified but not yet written to the database.

## **Update Method Applies To**

SSDBGrid

## UpdateError Event

[See Also](#)

[Applies To](#)

### Description

Occurs when an unexpected error occurs in a field when the control is updating the row.

### Syntax

```
Sub control_UpdateError (ColIndex As Integer, Text As String, ErrCode As Integer,  
ErrString As String, Cancel As Integer)
```

The event parameters are:

#### Parameter Description

---

<i>ColIndex</i>	The column of the current row that contains the error.
<i>Text</i>	The erroneous text.
<i>ErrCode</i>	The error code. The code represents a standard host environment error number.
<i>ErrString</i>	The error string. The string represents the actual error message.
<i>Cancel</i>	Set this to <b>True</b> to cancel any further updating of the row.

### Remarks

This event will be fired when moving off of the current row for each cell that does not pass validation whether in Bound, Unbound, or AddItem mode.

For more information on how to handle data-related errors, see "[How the Data Grid handles data validation and error checking.](#)"



## **UpdateError Event Applies To**

SSDBGrid

## **UpdateError Event See Also**

[IsCellValid](#)

[How the Data Grid handles data validation and error checking](#)

## Updating Rows from a Modal Form

There is a caveat when using a bound control to update a row or rows in a record set of a data control on a modal form. If a row is updated with an invalid field, such as a null key field, Visual Basic does not display an error until the modal form is hidden or unloaded. To overcome this Visual Basic limitation, include the following code in response to the **Error** event of the Visual Basic data control:

```
Sub Data1_Error (DataErr As Integer, Response As Integer)
    On Error Resume Next
    If DataErr Then
        Beep
        MsgBox Error (DataErr)
        DataErr = 0
        Response = 0
    End If
End Sub
```

**Note** This is applicable to any bound control including the standard Visual Basic controls.

## UseDefaults Property

Applies To

### Description

Determines whether the grid will use default values to populate new records.

### Syntax

*object* . **UseDefaults**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether new records will be populated with default values, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) The control will retrieve default values from the data source and use them to populate new records.
<b>False</b>	The control will not retrieve or use default values.

### Remarks

Retrieving default values from the data source affects the overall performance of the grid, and may cause problems with some systems, particularly ODBC.

If you are using the control with an ODBC data source, or you do not wish to populate new records with default values, you can set **UseDefaults** to False. Setting this property to False should also improve the performance of the control when adding records.

If you experience problems when using an ODBC data source, set this property to False.

## **UseDefaults Property Applies To**

SSDBGrid

## UseExactRowCount Property

Applies To

### Description

Determines whether the grid portion scans for the exact row count.

### Syntax

*object* . **UseExactRowCount**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the exact row count is scanned, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) Exact row count will be used.
<b>False</b>	Estimated row count will be used.

### Remarks:

When set to **True**, the control gets the last row in the record set when it initially binds to a data control. This may cause some performance penalties when accessing large databases.

## **UseExactRowCount Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## Using Data Widgets

### How Data Widgets Are Supplied

Data Widgets ships a total of four OCX files, grouped in pairs by 16-bit and 32-bit controls.

Filename	Controls Contained
SSDATA16.OCX	16-bit version of the Enhanced Data Control, DataOptionSet, and Data Command Button controls.
SSDATB16.OCX	16-bit version of the Data Grid, Data DropDown, and Data Combo controls.
SSDATA32.OCX	32-bit version of the Enhanced Data Control, DataOptionSet, and Data Command Button controls.
SSDATB32.OCX	32-bit version of the Data Grid, Data DropDown, and Data Combo controls.

### Including Data Widgets in Your Project

Custom controls are generally installed on a project-need basis. Once you have included a custom control in a project and saved that project, the control will be available whenever you subsequently open the project.

The method you use to add a Data Widgets control to your project varies depending on which programming environment you are using. Data Widgets comes in two varieties:

- § 32-bit OCX controls that are compatible with any programming system supporting OLE custom controls, supports advanced data binding, and runs in a 32-bit environment such as Windows NT or Windows 95.
- § 16-bit OCX controls that are compatible with any programming system supporting OLE custom controls and advanced data binding.

### Visual Basic 4.0

The use of OLE Custom Controls are new to Version 4.0 of Visual Basic, replacing the previously used VBX format. To use the Data Widgets controls in Visual Basic 4.0:

1. Open the project you want to add the Data Widgets control to.
2. Select **Custom Controls** from the *Tools* menu.
3. Select the appropriate Data Widgets control from the list of Available Controls. All of the Data Widget controls are prefaced by the name Sheridan (i.e., Sheridan Data Command Control). A checked box next to the control indicates that it has been selected.
4. Click the **OK** button.  
The control(s) you have selected are now added to your project.

### Other Languages

Data Widgets OCX controls are supported by a variety of host environments. To use Data Widgets in programming environments other than Visual Basic 4.0, consult your development tool's documentation for information on how to use *OLE Custom Controls* or *OCX Controls*.

Once the control is loaded, it should appear as an extension of your environment. Use the



control's [Property Pages](#) or the environment's property sheet (if available) to set up the control.

For more information on compatibility between different versions of the controls across different host environments, see [Introduction to OCX Controls](#).

## Using a Data DropDown in a Data Grid Column

Another powerful feature of the Data Grid is the ability to link a Data DropDown control to a column in the Data Grid. Similar to the cell button feature, this feature allows the user to click a button in the cell to drop down a list of choices.

The Data DropDown control can be bound to another record set in another data control. For instance, if one of the columns in the Data Grid contains a State Code, you can link in a Data DropDown and bind it to a data control with a list of State Codes and descriptions. When a button is clicked, a list of states would drop down for the user to choose from.

This is done by setting the **DropDownHwnd** property to the window handle of a Data DropDown control that is on your form. For more information on how to do this, refer to the Data DropDown.

## Using the Cell Button Feature of the Data Grid

Each column in the Data Grid allows you to include on the right of each cell a button for you to perform additional processing when pressed. To activate the cell button feature, set the property Style to **Edit Button** for the corresponding column. Whenever the button is clicked, the BtnClick event will be triggered, allowing you to perform any function you wish, such as displaying a dialog with a larger text box for memo-type fields.

## Using the Data Grid as a List Box

By default, the Data Grid does not look nor act much like a standard Windows list box. However, by setting just a few properties, the Data Grid can look and behave just like one. It can be used as a bound list box or an unbound virtual list.

To make the Data Grid work like a listbox, set the **AllowAddNew**, **AllowDelete**, **AllowDragDrop**, **AllowUpdate**, and **RecordSelectors** properties all to **False**. Set **SelectByCell** to **True**, **SelectTypeRow** to either **Single** or **MultiSelect**, and set **SelectTypeCol** to **None**. The Data Grid can now be used as a multi-column list box with optional headings. You can modify other properties as needed to customize the grid to your liking.

## VC++ ActiveCell Object example

Here is an example of how to get the active cell object from the SSDBGrid. You can use the wrapper class to automatically return the ActiveCell Object. This is a very common exchange using OLE controls in Visual C++; you use an empty variant to get the dispatch pointer to the collection, and then use that pointer to create the object.

```
// Uses the active cell object to display the text from the active
// cell in the Edit control on the dialog
//The ActiveCell method of the grid returns
//a dispatch pointer
LPDISPATCH pdispActiveCell = m_grid.ActiveCell();
ASSERT(pdispActiveCell);
//Use the dipatch pointer that was returned
//to create an instance of the ISSActiveCell Class
ISSActiveCell pobjCell(pdispActiveCell);
ASSERT(pobjCell);
//You can now use any of the ISSActiveCell methods
CString csText = pobjCell.GetText();
```

How can you get the text in a cell if the cell is not in the current row?

How should I validate data being entered into the grid?

## VC++ Common Questions Answers

### How can you get the text in a cell if the cell is not in the current row?

```
void CBndGridView::OnButton8()
{
    CSSDBGrid* pGrid = (CSSDBGrid*) GetDlgItem(IDC_SSDBGRIDCTRL1);
    COleVariant bookmark;    //The CellText() method needs a bookmark

    //initialize a variant for the column object index. We will look at
    columns 2
    //you can also use COleVariant instead of a standard variant
    //used standard variant for this example
    VARIANT vaCol;
    VariantInit(&vaCol);
    vaCol.vt=VT_I2;
    vaCol.iVal = 2;

    COleVariant va; //need an empty variant to initialize the collection

    LPDISPATCH pDispCols=NULL;
    pDispCols = pGrid->_Columns(va);

    //Get the columns collection by passing the LPDISPATCH to the
    //constructor
    ISSColumns Cols(pDispCols);

    LPDISPATCH pDispCol=Cols._Item(vaCol);
    //create the column object in the same way
    ISSColumn Col(pDispCol);

    //Get a bookmark for visible row number 5
    bookmark = pGrid->RowBookmark(5);
    //use the CellText method of the Column object to display the text
    AfxMessageBox( Col.CellText(bookmark) );

    VariantClear(&vaCol);

}
```

### How should I validate data being entered into the grid?

For example, if you wanted to ensure that the user always enters data in a cell, use the **BeforeColUpdate** event.

```
void CBndGridView::OnBeforeColUpdateSsdbgridctrl1(short ColIndex, const
VARIANT FAR& OldValue, short FAR* Cancel)
{
    CSSDBGrid* pGrid = (CSSDBGrid*) GetDlgItem(IDC_SSDBGRIDCTRL1);
    COleVariant va;
    COleVariant vIndex(ColIndex, VT_I2);
```

```
//Access the columns collection using an empty variant
LPDISPATCH pdispCols = pGrid->_Columns(va);
ISSColumns Cols(pdispCols);

//Access the particular column using the _Item() function of the columns
//collection
LPDISPATCH pdispCol = Cols._Item(vIndex);
ISSColumn ThisCol(pdispCol);

if ((ThisCol.GetText()).IsEmpty()) //GetText returns a CString
{ AfxMessageBox("This Column must be filled - Try again");
  *Cancel = 1; //This cancels the column update
  //You can also restore the old value...
  ThisCol.SetText((LPCSTR)OldValue.bstrVal);
}
}
```



## VC++ Create StyleSet example

Here is an example of how to create a StyleSet. You can use the wrapper class to create the StyleSet for you.

```
void CDataView::OnBtnStyleSets()
{
    CSSDBGrid* pGrid = (CSSDBGrid*) GetDlgItem(IDC_SSDBGRIDCTRL1);
    ASSERT(pGrid);

    LPDISPATCH    lpDispStyles = NULL;
    LPDISPATCH    lpDispStyle = NULL;
    ISSStyleSets* pStyles = NULL;
    ISSStyleSet*  pStyle=NULL;
    CString       StyleName;

    //use empty variant for collection
    VARIANT va;
    VariantInit(&va);
    //get the dispatch pointer to StyleSets Collection
    lpDispStyles = pGrid->StyleSets(va);
    if(lpDispStyles)
    {
        pStyles = new ISSStyleSets(lpDispStyles);

        if (pStyles)
        {
            COleVariant vaColor1((long)RGB(255,100,100), VT_I4);

            try
            {
                pStyles->Add("Color");
            }
            catch( CException* e)
            {
                AfxMessageBox("You can only create the StyleSet color
once!");
                e->Delete();
            }

            COleVariant va2((long)(pStyles->GetCount() -1), VT_I4);
            lpDispStyle = pStyles->GetItem(va2);

            if(lpDispStyle)
            {
                pStyle = new ISSStyleSet(lpDispStyle);
                pStyle->SetBackColor(vaColor1);

                CSSDBGrid* pGrid = (CSSDBGrid*) GetDlgItem(IDC_SSDBGRIDCTRL1);

                VARIANT vaCol;
                VariantInit(&vaCol);
                vaCol.vt=VT_I2;
                vaCol.iVal = 0;
            }
        }
    }
}
```

```

        ColeVariant var; //need an empty variant to initialize the
collection
        LPDISPATCH pDispCols=NULL;
        pDispCols = pGrid->Columns(var);
        ASSERT(pDispCols);
        //Get the columns collection by passing the LPDISPATCH to the
constructor
        ISSColumns Cols(pDispCols);
        ASSERT(Cols);
        LPDISPATCH pDispCol=Cols.Item(vaCol);
        ASSERT(pDispCol);
        //create the column object in the same way
        ISSColumn Col(pDispCol);
        ASSERT(Col);

        VARIANT row;
        VariantInit(&row);
        row.vt=VT_I2;
        row.iVal = 2;
        Col.CellStyleSet("Color",row );
        if(pStyles)
        {
            delete pStyles;
        }
        if(pStyle)
        {
            delete pStyle;
        }
    }
}
}

```

## VC++ Create StyleSet with Wrapper class example

Here is an example that shows you how to use the wrapper class to create the styleset for you.

```
//Create the color and the name of the style set
//you would like to create
COleVariant color((long)RGB(0,200,0), VT_I4);
CString name = "Green";
//Use the CGridHelper Class to create the styleset
//object for you, pass it the name of the Styleset
ISSStyleSet* mystyle = mygrid.CreateStyleSets(name);

//Use the newly create styleset
AfxMessageBox(mystyle->GetName());
//Set the BackColor of the styleset
mystyle->SetBackColor(color);
COleVariant row((short)1, VT_I2);
//Use the CGridHelper Class to get column
ISSColumn* Col =mygrid.GetColumn(0);
ASSERT(Col);
//Use the CellStyleSet method to apply a
//styleset to a certain column, and row
Col->CellStyleSet(name, row);
```

Once you have created the styleset you can then use it again referencing it by name.

```
CSSDBGrid* pGrid = (CSSDBGrid*)GetDlgItem(IDC_SSDBGRIDCTRL1);
mygrid.Create(pGrid);

COleVariant row(((short) pGrid->GetRow()), VT_I2);
ISSColumn* pCol = mygrid.GetColumn(2);
ASSERT(pCol);
if(pCol != 0)
{
    pCol->CellStyleSet("Green", row);
}
```

## VC++ CreateStyleSets Method

```
// #N CreateStyleSets
//
// #D Description:
// Returns a styleset pointer
//
// #A Arguments:
// #1 CString name - The name of the styleset you want to create
//
// #R Return:
// If successful returns a styleset pointer
// If unsuccessful returns 0
//
// #C Comments:
//
```

## VC++ Get SelBookmarks with Wrapper example

This example shows you how to use the wrapper class to get the SelBookmarks Collection.

```
// Use the CGridHelper Class to get the SelBookmarks Collection
CSSDBGGrid* pGrid = (CSSDBGGrid*)GetDlgItem(IDC_SSDBGGRIDCTRL1);
mygrid.Create(pGrid);

ISSelBookmarks* pSelBooks = mygrid.GetSelBookmarks();
ASSERT(pSelBooks);
if(pSelBooks == NULL)
{
    TRACE("Failed to get SelBookmarks pointer");
    return;
}
long num = pSelBooks->GetCount();
TRACE("The number of Selected Rows is %d", num);
```

You use the CellText method to get the text of a column that you are not currently on. You can not set the text this way, you need to be on the current row to actually change the text, but using CellText, or CellValue you can retrieve it.

```
COleVariant bookmark; //The CellText() method needs a bookmark
bookmark = pGrid->RowBookmark(5);
//use the CellText method of the Column object
AfxMessageBox( Col.CellText(bookmark) );
```

## VC++ GetActiveCell Method

```
// #N GetActiveCell
//
// #D Description:
//     Returns an activecell pointer
//
// #A Arguments:
//
//
// #R Return:
//     If successful returns the activecell pointer
//     If unsuccessful returns 0
//
// #C Comments:
//
```

## VC++ GetColumn Method

```
// #N GetColumn
//
// #D Description:
//     Returns a column pointer from an index number
//
// #A Arguments:
// #1     short Colindex - The zero based index of the column you want
//
//
// #R Return:
//     If successful returns a column pointer
//     If unsuccessful returns 0
//
// #C Comments:
//     Will Validate if column is valid to the grid you are using
//
```

## VC++ GetGroup Method

```
// #N  GetGroup
//
// #D  Description:
//      Returns a group pointer from an index number
//
// #A  Arguments:
// #1      short GroupIndex - The zero based index of the group you want
//
//
// #R  Return:
//      If successful returns a group pointer
//      If unsuccessful returns 0
//
// #C Comments:
//      Will Validate if Group is valid to the grid you are using
//
```



## **VC++ GetSelBookmarks Method**

```
// #N GetSelBookmarks
//
// #D Description:
//     Returns a SelBookmarks pointer
//
// #A Arguments:
//
//
// #R Return:
//     If successful returns a styleset pointer
//     If unsuccessful returns 0
//
// #C Comments:
//
```

[◀ Back](#)

## VC++ Wrapper Methods

**C**

[CreateStyleSets](#)

**G**

[GetActiveCell](#)

[GetColumn](#)

[GetGroup](#)

[GetSelBookmarks](#)

## **ValidateList Event Applies To**

SSDBCombo

SSBBDropDown

## ValidateList Event

Applies To

### Description

Occurs when the control needs to validate the data entered by the user against the list of values in the dropdown.

### Syntax

**Sub** control\_ValidateList (*Text* **As String**, *RtnPassed* **As Integer**)

<b>Part</b>	<b>Description</b>
-------------	--------------------

---

<i>Text</i>	The text to validate
-------------	----------------------

<i>RtnPassed</i>	Set this to notify the control that the data is valid or not. This parameter defaults to <b>False</b> .
------------------	---

### Remarks

If the control has the ListAutoValidate property set to **False**, the control will trigger this event when it needs to validate the data entered by the user.

When this event occurs, the *Text* parameter will contain the value to validate. If the value in *Text* is valid, set *RtnPassed* to **True**.

You should set the **ListAutoValidate** property to **False** and process this event to optimize list validation in large sets. Since the control cannot search a data control using index fields due to a Visual Basic limitation, using the data control 'Find' methods in this event can significantly improve validation time.

## Value Property (Bookmark Object)

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Sets or returns the value stored in a bookmark object.

### Syntax

*object* . **Value**[= *value*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>value</i>	A variant expression specifying the value stored in a bookmark.

## **Value Property (Bookmark) Applies To**

Bookmark object (SSDBData only)

**Value Property (Bookmark) See Also**

String

## Value Property (Button Object)

Applies To

### Description

Sets or returns the current state (checked / not checked) of the button object.

### Syntax

*object* . **Value**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>value</i>	A boolean expression specifying whether the button is selected or not.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	(Default) Button object is selected.
<b>False</b>	Button object is not selected.



## **Value Property (Button Object) Applies To**

Button object

## Value Property (SSDBGrid)

[See Also](#)

[Applies To](#)

### Description

Returns the value stored in a column.

### Syntax

*object* . **Value**[= *value*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Value</i>	A variant expression specifying the value stored in a column.

### Remarks

**Value** returns the contents of a column in its native Variant form while **Text** returns the contents of a column as a string.

## **Value Property Applies To**

ActiveCell Object

Column Object

ssRowBuffer Object

## **Value Property See Also**

Text

## VertScrollBar Property

Applies To

### Description

Determines whether a vertical scrollbar is displayed in a column.

### Syntax

*object* . **VertScrollBar**[= *boolean* ]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether a vertical scrollbar is displayed, as described in Settings.

### Settings

<b>Setting</b>	<b>Description</b>
<b>True</b>	Vertical scrollbar is displayed.
<b>False</b>	(Default) Vertical scrollbar is not displayed.

### Remarks:

The vertical scrollbar is useful when there is data that extends past the width of the column. The scrollbar allows the user to scroll through the entire contents of the cell.

## **VertScrollBar Property Applies To**

Column object

## VisibleCols Property

[See Also](#)

[Applies To](#)

### Description

Returns the number of visible columns in the grid.

### Syntax

*object* . **VisibleCols**

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.

### Remarks

**VisibleCols** is only available at runtime and is read-only. **VisibleCols** returns an integer expression specifying the number of visible columns in the grid.

## **VisibleCols Property Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid



## **VisibleCols See Also**

Col

Cols

## **VisibleGrps Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## VisibleGrps Property

Applies To

### Description

Returns the number of visible groups in the grid.

### Syntax

*object* . **VisibleGrps**

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.

### Remarks

**VisibleGrps** is only available at runtime and is read-only. **VisibleGrps** returns an integer expression specifying the number of visible groups in the grid.

## **VisibleRows Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## VisibleRows Property

Applies To

### Description

Returns the number of visible rows in the grid.

### Syntax

*object* . **VisibleRows**[= *number*]

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the number of visible rows in the grid.

### Remarks

**VisibleRows** is only available at runtime and is read-only.

## Visual C++ Information

### Methods Included in the Wrapper Class

### Inserting controls into your VC++ project

### Other VC++ Examples

### Important Note on Using Examples

### Some Common Questions

Data Widgets 2.0 ships with a number of samples that illustrate how to use the controls in Microsoft Visual C++ version 4.0 or later. These samples are included in the SAMPLES\MSVC directory under your installation directory. The samples include a wrapper class that exposes the functionality of the controls in VC++.

The wrapper class is designed to encapsulate the use of some of the functions of the Sheridan Software SSDBGrid controls. The source code to the CGridHelper class is contained in GridHelper.cpp and GridHelper.h.

The class holds a pointer to the control you are using when you pass it through the create function. Remember, if you use GetDlgItem it returns a temporary pointer only, so you would need to call the create function again the next time you need to use it. There is some sample code that shows you how to get a pointer you can store.

You can obtain a permanent pointer to the Grid on a dialog by using this code. Please note that this will give you an ASSERT failure if you also have a member variable defined for the grid in ClassWizard.


```
CSSDBGrid* pGrid2 = new CSSDBGrid();
pGrid2->Attach(GetDlgItem(IDC_SSDBGRIDCTRL1)->GetSafeHwnd());

TRACE("The caption is %s\n", pGrid2->GetCaption());
pGrid2->Detach();
delete pGrid2;
```

The wrapper class assists in using the collections that are part of the controls. If the function fails its return value is 0 (zero), so that you can use the ASSERT() macro in your code to check for a successful result.

In the example of the wrapper classes GetColumn() function, you pass GetColumn the index of the column you want to use. For example 1. You would then receive a pointer to that column if the function is successful, and zero if the function fails. You can then use this pointer to call any of the methods or get or set any of the properties of the column object. You can store the pointer that was returned to you for later use if you want, but you are responsible for deleting the object.

## **What are Bookmarks? (Enhanced Data Control)**

Bookmarks are a powerful feature that allow you to "flag" a record you want to remember. The EDC allows you to Add, Store, and Delete bookmarks without the need for coding. To access a stored bookmark, the user need only click on the  button, where a dropdown list of stored bookmarks will appear for selection.

## [◀ Back](#) **What is Data Widgets?**

Data Widgets is a set of custom controls that allow you to design front-ends for database applications with all the simplicity and power you have come to expect from your host development application.

Designed with ease of use in mind, Data Widgets virtually eliminates the need for time-consuming coding when developing applications involving database operations. What used to take hours of development can now take minutes. All you need to do is drop a control on a form, set a few properties, and Data Widgets does the rest!

Data Widgets includes six bound custom controls, each for specific data-manipulation functions, provided in both 16-bit and 32-bit OLE Custom Control (OCX) format.

[Data Widgets Features](#)

[Using Data Widgets](#)

[Introduction to OCX controls](#)

[Optimizing Data Widgets](#)

[StyleSets](#)

[Property Pages](#)



New! **2.0**

## Data What's New?

Perhaps the most significant change in Data Widgets 2.0 is the transition from VBX to OCX format for custom controls. The OCX format utilizes Microsoft's OLE automation specifications. Controls such as Data Widgets can now be used on a range of development environments, whereas in the past, Data Widgets was limited to environments that supported the VBX format. For a discussion of OCX related issues, refer to the section **Introduction to OCX Controls**.

Version 2.0 of Data Widgets introduces the use of objects and collections. Objects allow you to manipulate the custom controls much more easily and with more power than in the past. For example, you can easily set a property specific to an object (such as an individual column) by accessing the object directly. This means that you can customize Data Widgets to suit your individual needs. Collections are simply organized groupings of objects. For more information on objects and collections, see the section entitled **Object Concepts**.

With this release of Data Widgets also comes the transition from accessing and manipulating grid rows by row number to using bookmarks. Row numbers are still used in Data Widgets 2.0, but their meaning has been redefined.

There are now two types of row numbers, *Display* row numbers which indicate the row number as it appears in the current view, and *Absolute* row numbers which indicate the row number relative to the entire grid. Absolute row numbers are *only* used in AddItem mode, while display row numbers are used in bound, unbound, and AddItem modes of the grid.

Because of the many changes, code that worked in Version 1.0 will need to be modified to take advantage of the new structure. Additionally, many of the properties that existed in 1.0 have either been altered to conform with this new structure, or eliminated completely. To help ease the transition of users from 1.0 to 2.0, refer to the Online Help file 'DATW1TO2.HLP'.

## **WhereIs Applies To**

SSDBData

SSDBOptSet

SSDBGrid

## WhereIs Method

[See Also](#)

[Applies To](#)

[Example](#)

### Description

Returns the current area of the control to which the cursor is pointing. Returns values for nothing, button areas, and caption area.

### Syntax

*object* . **WhereIs**(*X* **As Single**, *Y* **As Single**, [*scale* **As Variant**])

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>X</i>	Determines the X coordinate.
<i>Y</i>	Determines the Y coordinate.
<i>scale</i>	(Optional) Determines the scale to use as described in Settings.

### Settings

The settings for *scale* are:

<b>Setting</b>	<b>Description</b>
0	Twips
1	Pixels
2	Container Coordinates
3	HiMetric

There are [constants](#) for the settings of this parameter.

### Remarks

For **SSDBData**, **WhereIs** returns the following values:

<b>Value</b>	<b>Description</b>
0	Pointing at nothing
1	Pointing at caption
2	Pointing at bevel
3	Pointing at button "First"
4	Pointing at button "Last"
5	Pointing at button "Previous Page"
6	Pointing at button "Next Page"
7	Pointing at button "Previous Record"
8	Pointing at button "Next Record"
9	Pointing at button "Add Record"
10	Pointing at button "Cancel Add Record"

- 11            Pointing at button "Update Record"
- 12            Pointing at button "Delete Record"
- 13            Pointing at button "Find Next"
- 14            Pointing at button "Find Previous"
- 15            Pointing at button "Find"
- 16            Pointing at button "Add Bookmark"
- 17            Pointing at button "Clear Bookmarks"
- 18            Pointing at button "Goto Bookmark"

There are constants for the settings of this parameter.

For **SSDBGrid, WhereIs** returns the following values:

<b>Value</b>	<b>Description</b>
0	Pointing at nothing
1	Pointing at Grid Heading
2	Pointing at Group Header
3	Pointing at Column Header
4	Pointing at Grid Selector
5	Pointing at Record Selector
6	Pointing at Background
7	Pointing at Data

There are constants for the settings of this parameter.

For **SSDBOptSet, WhereIs** returns the following values:

<b>Value</b>	<b>Description</b>
0	Pointing at nothing
1	Pointing at button
2	Pointing at caption

There are constants for the settings of this parameter.

## **WhereIs Method See Also**

[ButtonFromCaption](#)

[ButtonFromPos](#)

## WidthGap Property

See Also [\\_\\_\\_\\_\\_](#) Applies To

### Description

Returns or sets the minimum horizontal distance between columns.

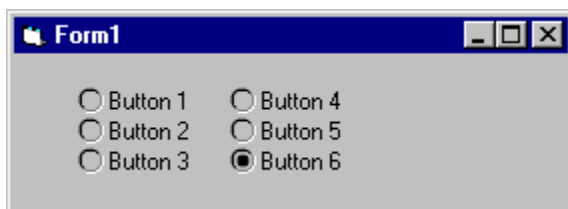
### Syntax

*object* . **WidthGap**[= *number*]

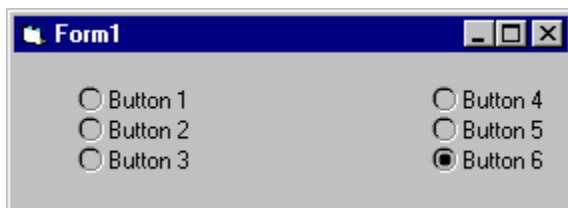
Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the minimum amount of horizontal distance between columns.

### Remarks

The following example shows a DataOptionSet with a small **WidthGap** setting:



The following example shows a DataOptionSet with a large **WidthGap**:



## **WidthGap Property Applies To**

SSDBOptSet

## **WidthGap Property See Also**

[ColOffset](#)

[RowOffset](#)



## **WordWrap Property Applies To**

SSDBCommand

SSDBOptSet

SSDBData

## WordWrap Property

[See Also](#)

[Applies To](#)

### Description

Determines if the text specified in the **Caption** property will be wrapped.

### Syntax

*object* . **WordWrap**[= *boolean*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A Boolean expression specifying whether to wrap the caption text, as described in Settings.

Settings	Description
<b>True</b>	(Default) Caption text will wrap to multiple lines as needed.
<b>False</b>	Caption text will remain on a single line.

### Remarks

Word wrapping causes text to be split into multiple lines if the caption width exceeds the width of the control, or in the case of SSDBOptSet, the width of the column.

The following is an example of a button without wordwrap:



The following is an example of a button with wordwrap:



**WordWrap Property See Also**

[MultiLine](#)

## Add Method example

This example adds the current row bookmark to a selected bookmarks collection:

```
SSDBGrid1.SelBookmarks.Add(SSDBGrid1.Bookmark)
```

This example adds five buttons to a button collection:

```
SSDBOptSet1.Buttons.Add 5
```

This example adds a second and third column to a columns collection:

```
SSDBGrid1.Columns.Add 1
```

```
SSDBGrid1.Columns.Add 2
```

This example adds a second and third group to a groups collection:

```
SSDBGrid1.Groups.Add 1
```

```
SSDBGrid1.Groups.Add 2
```

### **AddItem Method example**

The following code adds two cells to an AddItem grid with the **FieldSeparator** property set to a comma:

```
SSDBGrid1.AddItem "Hello,World"
```

### **AddItem Method (Column Object) example**

The following example adds the state abbreviation NY to a combo box:

```
SSDBGrid1.Columns(3).AddItem "NY"
```

### **AfterClick Event example**

The following code displays a dialog box after the user has performed a database function:

```
Private Sub SSDBCommand1_AfterClick()  
    MsgBox ("Database Action Performed!")  
End Sub
```

### **AfterDelete Event example**

This example shows how to use the AfterDelete event to execute custom code:

```
Private Sub SSDBCommand1_AfterDelete()  
    MsgBox ("Database Record Deleted!")  
End Sub
```



### **AfterInsert Event example**

This example shows how to use the AfterInsert event to execute custom code:

```
Private Sub SSDBCommand1_AfterInsert()  
    MsgBox ("New Record Inserted!")  
End Sub
```

## **AfterUpdate Event example**

This example shows how to use the AfterUpdate event to execute custom code:

```
Private Sub SSDBCommand1_AfterUpdate()  
    MsgBox ("Record Updated!")  
End Sub
```

## BackColor Property example

The following is an example of how the **BackColor** property affects the SSDBOptSet control:

```
SSDBOptSet1.Buttons(1).BackColor = &H00808080& ' Set to dark gray
```



### **BeforeColUpdate Event example**

This example demonstrates how to restore the old value:

```
Cancel = 0  
SSDBGrid1.Columns(ColIndex).Value = OldValue
```

## **Bold Property example**

This sample code sets the caption text for the control to bold:

```
SSDBOptSet1.Caption = "DataOptionSet Example!"  
SSDBOptSet1.Font.Bold = True  
SSDBGrid.Headfont.Bold = True
```

## Bookmark Property example

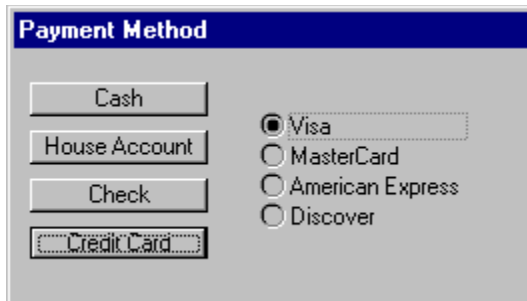
This sample code uses the Bookmark property of the RowBuffer object to assign a unique bookmark:

```
For j = 0 To SSDBGrid1.Cols - 1           'For each column in the
grid                                     'grid
    RowBuf.Value(i, j) = myarray(p, j)   'Set the value of
each column in the row buffer to the corresponding value in the array
Next                                     'Set the value of the
RowBuf.Bookmark(i) = p                  'set the value of the
bookmark for the current row in the rowbuffer
```

Also see the [example code](#) for the **UnboundReadData** event.

## ButtonEnabled Property example

The following example demonstrates how the **ButtonEnabled** property can be used. In our Payment Method example below, the four credit card options are set with an initial property of **ButtonEnabled=False**. Only when the user clicks on the "Credit Card" button do we want the various credit card types enabled for selection.



The following code is placed in the **Click** event of the "Credit Card" button:

```
SSDBoptSet1.Buttons(0).Enabled = True  
SSDBoptSet1.Buttons(1).Enabled = True  
SSDBoptSet1.Buttons(2).Enabled = True  
SSDBoptSet1.Buttons(3).Enabled = True
```

### **ButtonFromCaption Method example**

The following example looks for a button with the caption of "Visa" and hides it:

```
SSDBOptSet1.ButtonFromCaption ("Visa").Visible = False
```



## ButtonFromPos Method example

This sample looks for a button that resides at (100,100) and changes its position:

```
Sub SSDBOptSet1_MouseMove (Button As Integer, Shift As Integer, X As  
Single, Y As Single)  
  
SSDBOptSet1.ButtonFromPos (100,100).ColOffset = 10  
SSDBOptSet1.ButtonFromPos (100,100).RowOffset = -15  
  
End Sub
```

## ButtonVisible Property example

This example demonstrates the use of the ButtonVisible property. In this example, the first button in a group is selected using the IndexSelected property, then the visibility of the button is set using the **ButtonVisible** property:

```
SSDBOptSet1.IndexSelected = 0  
SSDBOptSet1.ButtonVisible = False
```

Similarly, you can use the **Visible** property of the Button Object to display or hide individual buttons. Here the second Button object in the Buttons Collection is made invisible:

## Buttons Collection example

To refer to a button within a collection, use the following syntax:

```
Control.Buttons(Button Number).Property = Value
```

The following example demonstrates various properties that can be set:

```
SSDBOptSet1.Buttons(0).Caption = "This is button #1 of the collection"  
SSDBOptSet1.Buttons(1).Caption = "This is button #2 of the collection"  
SSDBOptSet1.Buttons(2).RowOffset = 5
```

The following example demonstrates adding four buttons to a SSDBOptSet control:

```
SSDBOptSet1.Buttons.Add(4)
```

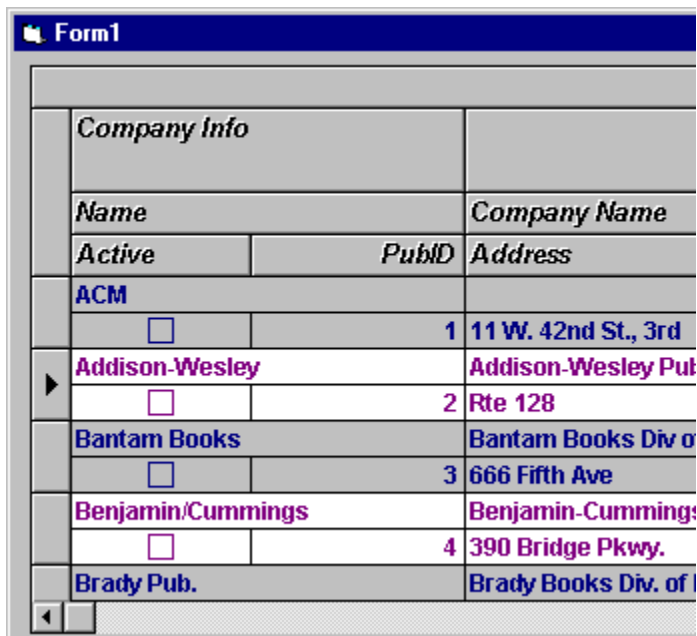
## Caption Property example

This sample code sets the caption text of a SSDBCommand button:

```
SSDBCommand1.Caption = "Next Student"
```

This sample code sets the caption text for a **Column** object and a **Group** object header:

```
SSDBGrid1.Columns(1).Caption = "Active"  
SSDBGrid1.Groups(0).Caption = "Company Info"
```



<i>Company Info</i>		
<i>Name</i>	<i>Company Name</i>	
<i>Active</i>	<i>PubID</i>	<i>Address</i>
<b>ACM</b>		
<input type="checkbox"/>	1	11 W. 42nd St., 3rd
<b>Addison-Wesley</b>		
<input type="checkbox"/>	2	Rte 128
<b>Bantam Books</b>		
<input type="checkbox"/>	3	666 Fifth Ave
<b>Benjamin/Cummings</b>		
<input type="checkbox"/>	4	390 Bridge Pkwy.
<b>Brady Pub.</b>		
Brady Books Div. of I		

This sample code sets the caption text of a group of DataOptionSet buttons. When using the caption property with the DataOptionSet, only the selected button is affected.

```
SSDBOptSet1.Buttons(0).Caption = "Visa"  
SSDBOptSet1.Buttons(1).Caption = "Mastercard"  
SSDBOptSet1.Buttons(2).Caption = "AMEX"
```

### **CellText Method example**

The following example takes the text from a cell and displays it in a message box:

```
DispText = SSDBGrid1.Columns(2).CellText(SSDBGrid1.LastRow)
MsgBox ("The value for that Row is :" + DispText)
```

### **CellValue Method example**

The following example uses the value of a cell to perform a calculation:

```
SalesTax = SSDBGrid1.Columns(5).CellValue(SSDBGrid1.Bookmark)
Total = (SubTotal * SalesTax)
```

## Find Example

It is possible to use the **CloseFindDialog** event in combination with the **Find** method to implement custom code that is invoked when the user performs a search, and yet still uses the built-in search features of the control.

For example, suppose you wish to log each search performed by the user. You might write a subroutine called LogQuery to write the criteria of each query to a log file. Your subroutine would be placed in the **CloseFindDialog** event. After calling your subroutine to log the query, you would use the **Find** method to actually perform the query:

```
Sub SSDBData1_CloseFindDialog(FindString as Variant, Criteria As  
Variant, Direction As Variant, ColToSearch As Variant, Cancel As  
Integer)  
  
    Cancel = True  
    LogQuery FindString, Criteria, Direction, ColToSearch  
    Find FindString, Criteria, Direction, ColToSearch  
  
End Sub
```

Note that it is important to set the value of *Cancel* to **True**. Otherwise, the query will be performed a second time after exiting the event.

## ColChanged Property Example

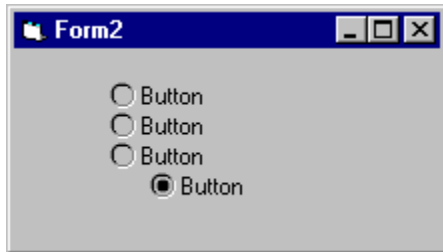
The following example demonstrates use of the **ColChanged** property

```
If SSDBGrid1.Columns(0).ColChanged Then
    MsgBox ("The name has changed!")
EndIf
```



### ColOffset Property example

The following example demonstrates the effect of the ColOffset property. The last button has a ColOffset value of 20:



The code for this example would be:

```
SSDBOptSet1.Buttons(3).ColOffset = 20
```

### **ColPosition Method example**

The following code returns the index for the column that appears fifth on the grid:

```
X=SSDBGrid1.ColPosition (4)
```

The following code returns the index for the column that appears second in the group:

```
X=SSDBGrid1.Groups (3) .ColPosition (1)
```

### **Columns Collection example**

To refer to a column within a collection, use the following syntax:

```
Control.Columns(Column Number).Property = Value
```

The following example demonstrates setting a property:

```
SSDBGrid1.Columns(0).Caption = "This is Column #1 of the collection"
```

## Count Property example

The following code takes the number of buttons and displays it in a message box:

```
Dim NumButtons as Integer
NumButtons = SSDBOptSet1.Buttons.Count
Msg = "Total Number of Buttons: " + Str(NumButtons)
Response = MsgBox(Msg, 0)
```

## **DropDownHwnd Property Example**

The following code ties the second column of a Data Grid to a Data DropDown:

```
SSDBGrid1.Columns(1).DropDownHwnd = SSDBDropDown1.Hwnd
```

### FieldDelimiter Property example

In the following example, the **FieldSeparator** is set to ",":

```
SSDBGrid1.FieldDelimiter = "!"  
SSDBGrid1.AddItem ("!Hello, world!,!How Are You?!")
```

This example adds two columns to the grid. If **FieldDelimiter** was set to "None", it would have added three because it would have interpreted the comma in "Hello, World" as a separator.

## FieldValue Property Example

The following example displays a message box containing the employee name corresponding to the current record, assuming that **DataField** is set to the field "EmployeeName":

```
MsgBox ("Current Employee: "+SSDBData1.FieldValue)
```

## Font Object example

The following code changes the **Bold** property setting of a **Font** object identified by the **Font** property of a Data Widgets control:

```
SSDBOptSet.Font.Bold = True ` Sets caption text to bold
```

The following code changes the **Size** property setting of a **Font** object identified by the **Font** property of a Data Widgets control:

```
SSDBData.Font.Size = 10 ` Set the font size to 10 points
```

The following code changes the **Name** property setting of a **Font** object identified by the **Font** property of a Data Widgets control:

```
SSDBCommand.Font.Name = "Arial" ` Sets the font to Arial
```



### Font3D Property example

This sample code sets the caption text to 3D with heavy shading if the font is greater than 18 points.

```
If (SSDBCommand1.Font.Size > 18) Then
    SSDBCommand1.Font3D = 2      ' Raised w/heavy shading
Else
    SSDBCommand1.Font3D = 1      ' Raised w/light shading
End If
```

## **GetBookmark Method example**

The following example moves the current row down two records:

```
BM=SSDBGrid1.GetBookmark(2)  
SSDBGrid1.Bookmark = BM
```

## Groups Collection example

To refer to a column within a groups collection, use the following syntax:

```
Control.Groups(Group Number).Property = Value
```

The following example demonstrates various properties that can be set:

```
SSDBGrid1.Groups(0).Caption = "This is Group #1 of the collection"  
SSDBGrid1.Groups(1).Caption = "This is Group #2 of the collection"
```

### **GrpHeadClick Event example**

The following example displays a message box when a group header is clicked, indicating the group number:

```
Sub SSDBGrid1_GrpHeadClick (ByVal GrpIndex As Integer)
    MsgBox "Grp" + Str$(GrpIndex)
End Sub
```

### **GrpPosition Method example**

The following code returns the index for the group that appears third on the grid:

```
X=SSDBGrid1.GrpPosition (2)
```

## HeadClick Event example

The following example displays a message box when a column header is clicked, indicating the column number:

```
Sub SSDBGrid1_HeadClick (ByVal ColIndex As Integer)
    MsgBox "Column :" + Str$(ColIndex)
End Sub
```

## HeadFont Object example

The following code changes the **Bold** property setting of a **HeadFont** object:

```
SSDBGrid.HeadFont.Bold = True ` Sets caption/header text to bold
```

## InitColumnProps Event example

The following example sets the initial caption alignment for a grid:

```
Private Sub SSDBGrid1_InitColumnProps()  
    SSDBGrid1.Groups(0).CaptionAlignment = 2 ' Sets Group 0 Center  
    SSDBGrid1.Groups(1).CaptionAlignment = 0 ' Sets Group 1 Left  
    SSDBGrid1.Groups(2).CaptionAlignment = 1 ' Sets Group 2 Right  
End Sub
```



## IsItemInList Method example

The following example displays an error if the text in the edit portion is not in the dropdown list:

```
Sub SSDBCombo1_LostFocus()  
    If Not SSDBCombo1.IsItemInList Then  
        MsgBox "Text Is Not In List!"  
    End If  
End Sub
```

### **IsValid Method example**

An example of when text would be invalid is if the text is null and **AllowNull** is false. You could then display an error message such as:

```
Sub SSDBCombo1_LostFocus()  
    If not SSDBCombo1.IsValid then  
        MsgBox "Text Is Not Valid!"  
    EndIf  
End Sub
```

## Italic Property example

This sample code sets the caption text for the control to italic:

```
SSDBOptSet1.Caption = "DataOptionSet Example!"  
SSDBOptSet1.Font.Italic = True  
  
SSDBGrid.Headfont.Italic = True
```

## List Property example

The following example demonstrates the **List** property by adding two strings:

```
SSDBGrid1.Columns(2).List(3) = "Hello"  
SSDBGrid1.Columns(2).List(4) = "World"
```

## Locked Property example

The following code demonstrates the **Locked** property by generating an error when a user attempts to edit a locked column:

```
Private Sub SSDBGrid1_Click()  
    If SSDBGrid1.Columns(currentcol).Locked = True Then  
        MsgBox ("This field can not be edited!")  
    End If  
End Sub
```

### **MinColWidth Property example**

The following example sets the minimum column width for the fourth button to 50:

```
SSDBOptSet1.Buttons(3).MinColWidth = 50
```

### **MinHeight Property example**

The following example sets the minimum height of the control to 250:

```
SSDBOptSet1.MinHeight = 250
```

## MousePointer Property example

The following sample code sets the mouse icon to the custom type using the icon "DISK.ICO":

```
SSDBOptSet1.MousePointer = 99  
SSDBOptSet1.MouseIcon = DISK.ICO
```



### **MoveRecords Method example**

The following code moves forward 10 records in a grid:

```
SSDBGrid1.MoveRecords (10)
```

The following code moves backward 5 records in a grid:

```
SSDBGrid1.MoveRecords (-5)
```

## NumberFormat Property example

The following examples demonstrate the effects **NumberFormat** has on data:

<b>Data</b>	<b>NumberFormat String</b>	<b>Displayed Result</b>
5459.4	\$ ##,##0.00	\$ 5,459.40
334.9	###0.000	334.900
100	Currency	\$100.00

## NumberOfButtons Property example

This example creates five buttons in the DataOptionSet and creates captions for them:

```
SSDBOptSet1.NumberOfButtons = 10
For X = 0 To (SSDBOptSet1.NumberOfButtons - 1)
    SSDBOptSet1.Buttons(x).Caption = "Button #" + STR$(X)
Next X
```

### OptionValue Property example

The following example shows three buttons which are bound to a database with the field "City". When the current field in the database matches one of the option values, the button becomes selected. If the user selects another option button, the field of the current database's record will be set to the button's option value.

```
SSDBOptSet1.Buttons(0).OptionValue = "NY"  
SSDBOptSet1.Buttons(1).OptionValue = "NJ"  
SSDBOptSet1.Buttons(2).OptionValue = "CT"
```

### **PageValue Property example**

This sample code sets the **PageValue** so that the control will move forward 40 records when they click the "Next Page" button.

```
SSDBData1.PageValue = 40  
SSDBCommand1.PageValue = 40
```

## Picture Property example

To load the picture at runtime you need to use the LoadPicture property as shown here:

```
SSDBCommand1.Picture = LoadPicture("icons\writing\erase02.ico")
```

## PictureButtons Property example

This sample code loads a custom **PictureButton** bitmap:

```
SSDBData.PictureButtons = LoadPicture ("c:\bitmaps\mybutns.bmp")
```

## ReadType Property Example

Unless you use the **ReadType** property in your code for the **UnboundReadData** event, your program may read data unnecessarily and performance may be adversely affected. This is particularly true in the DataDropDown and DataCombo controls, where the user is frequently searching for information in a specific field.

The following is based on the example code for the **UnboundReadData** event. For the full listing of the example code, see the [UnboundReadData example](#).

```
For i = 0 To RowBuf.RowCount - 1           'For each row in the row
buffer                                     'buffer
    If p < 0 Or p > counter Then Exit For   'If the pointer is
outside the grid then stop this           'outside the grid then stop this

        Select Case RowBuf.ReadType         'Optimize the data read
based on the ReadType                     'based on the ReadType

            Case 0 'Read All Data
                For j = 0 To SSDBGrid1.Cols - 1 'For each column
in the grid                               'in the grid
                    RowBuf.Value(i, j) = myarray(p, j) 'Set the value
of each column in the row buffer to the corresponding value in the
array                                     'of each column in the row buffer to the corresponding value in the
array
                Next
                RowBuf.Bookmark(i) = p       'set the value of
the bookmark for the current row in the rowbuffer 'the bookmark for the current row in the rowbuffer

            Case 1 'Read bookmarks only
                RowBuf.Bookmark(i) = p       'set the value of
the bookmark for the current row in the rowbuffer 'the bookmark for the current row in the rowbuffer

            Case 3 'Read bookmarks and bound column
                RowBuf.Value(i, 0) = myarray(p, 0) 'this just assumes
the first column is the bound column      'the first column is the bound column
                RowBuf.Bookmark(i) = p       'set the value of
the bookmark for the current row in the rowbuffer 'the bookmark for the current row in the rowbuffer

        End Select

        If ReadPriorRows Then               'move the pointer
forward or backward, depending _         'forward or backward, depending _
            p = p - 1                       'on which way it's
supposed to move                         'supposed to move
        Else
            p = p + 1
        End If

        r = r + 1                           'increment the number
of rows read                             'of rows read
    Next i
```



## ReBind Method example

The following example removes a column from a bound grid then replaces it by calling the **ReBind** method:

```
SSDBGrid1.Columns.Remove(2)  
SSDBGrid1.ReBind
```

## Redraw Property example

The following code demonstrates the **Redraw** property:

```
SSDBGrid1.Redraw = False
For X = 1 To 5000
    SSDBGrid1.AddItem "This is Row " + STR$(X)
Next X
SSDBGrid1.Redraw = True
```

## Remove Method example

The following example illustrates use of the remove method by deleting the fourth button in a collection of ten buttons:

```
SSDBOptSet1.Buttons(4).Caption = "Fifth"  
SSDBOptSet1.Buttons.Remove (3) ` Fourth button is removed  
X = SSDBOptSet1.Buttons(3).Caption ` X contains "Fifth" since all buttons  
shifted
```

## **RemoveAll Method (AddItem Mode) example**

The following code removes all strings from an AddItem grid:

```
SSDBGrid1.RemoveAll
```

## RemoveAll Method (Collections) example

The following example illustrates use of the **RemoveAll** method:

```
SSDBData1.Bookmarks.RemoveAll      'All bookmarks deleted  
SSDBOptSet1.Buttons.RemoveAll     'All buttons deleted  
SSDBGrid1.Columns.RemoveAll       'All columns deleted
```

### **RemoveAll Method (Column Object) example**

The following code example removes all items from a combo box in the third column:

```
SSDBGrid.Columns(2).RemoveAll
```

### **RemoveItem Method (AddItem Mode) example**

The following example removes the fifth row of an AddItem grid:

```
SSDBGrid1.RemoveItem (4)
```

### **RemoveItem Method (Column Object) example**

The following code example removes the fourth item of a combo box in the first column:

```
SSDBGrid1.Columns(0).RemoveItem(3)
```



## RowBookmark Method example

This example selects the last five visible rows in the grid:

```
VIS = (SSDBGrid1.VisibleRows - 6)
For X = VIS to SSDBGrid1.VisibleRows - 1
    SSDBGrid1.SelBookmarks.Add SSDBGrid1.RowBookmark(X)
Next X
```

**For examples of how to use the RowCount property, see:**

[DataGrid: Exercise 2 - Unbound Mode](#)

[UnboundReadData Event Example](#)

## RowLoaded Event example

The following example demonstrates how the **RowLoaded** event can be used to view and act upon data as it is read into the grid. In this scenario, if the State field contains "New York", a graphic of a heart is displayed in the cell.

```
SSDBGrid1.StyleSets("rowload1").BackColor = RGB(0, 128, 255)
    SSDBGrid1.StyleSets("rowload1").ForeColor = RGB(0, 0, 0)
    SSDBGrid1.StyleSets("rowload1").Font.Name = "Arial"
    SSDBGrid1.StyleSets("rowload1").Font.Strikethrough = True
    SSDBGrid1.StyleSets("rowload1").Picture = "e:\mary\graphics\
heart.bmp"

    If SSDBGrid1.Groups(1).Columns(1).CellText(Bookmark) = "New York"
Then
    SSDBGrid1.Groups(1).Columns(1).CellStyleSet "rowload1"
End If
```

### SavedBookmark Property example

Create two SSDBCommand buttons; one captioned "Save Bookmark" with **DatabaseAction** = 6, and the other captioned "Goto Bookmark" with **DatabaseAction** = 7.

In the **AfterClick** event for the Save Bookmark button, add the following code:

```
SSDBCommand2.SavedBookmark = SSDBCommand1.SavedBookmark
```

After scrolling through the database, you can click the "Goto Bookmark" button to return to the record whose bookmark you saved.

## SelBookmarks Collection example

The following example will add the first five rows to the collection:

```
Dim i as integer
SSDBGrid1.MoveFirst ' Position at the first row
for i = 0 to 4
    SSDBGrid1.SelBookmarks.Add SSDBGrid1.Bookmark
    SSDBGrid1.MoveNext
next i
```

It is also easy to access the rows in the **SelBookmarks** collection without moving the current row position. For example, if there was a column in the grid called Amount and you wanted to add up all the rows that were selected to get a total, you could use the following code:

```
Dim nTotal as long
Dim nTotalSelRows as integer
Dim i as integer
Dim bkmrk as Variant ' Bookmarks are always defined as variants

nTotalSelRows = SSDBGrid1.SelBookmarks.Count

' In the following, get the bookmark of the selected rows

for i = 0 to nTotalSelRows
    bkmrk = SSDBGrid1.SelBookmarks(i)
    nTotal = nTotal + SSDBGrid1.Columns("Amount").CellValue(bkmrk)
next i

Debug.Print "The total amount = " & Format(nTotal, "Currency")
```

## Size Property example

The following sample code sets the font size to 18 points:

```
SSDBOptSet.Font.Size = 18  
SSDBGrid.Headfont.Size = 18
```

## Strikethrough Property example

The following sample code causes the text to appear with strikethrough formatting:

```
SSDBOptSet1.Font.Strikethrough = True  
SSDBGrid.Headfont.Strikethrough = True
```

## String Property example

This sample stores a value to a bookmark and identifies it with the **String** property:

```
Dim vbookmark as variant
vbookmark = SSDBCommand1.SavedBookmark
SSDBData1.Bookmarks(0).Value = vbookmark
SSDBData1.Bookmarks(0).String = "The first bookmark"
```



### TagVariant Property example

The following code illustrates how the **TagVariant** property can be set to a double-precision floating point number:

```
Dim TaxRate As Double  
TaxRate = 0.825  
SSDBCommand1.TagVariant = TaxRate
```

### **TextFormat Property example**

In this example, data displayed in the edit portion of the SSDBCombo will be displayed as currency:

```
SSDBCombo1.TextFormat = "Currency"
```

## UnboundAddData Event Example

The following example shows how you can use the UnboundAddData event to take information from a row that has been added to the grid and append it to an array in memory. Also see [Data Grid: Exercise 2](#) for an example of how to use unbound mode with custom code to access a database.

First, place a DataGrid on a form in Visual Basic, and set it to Unbound mode. Then enter the following code:

### In the (declarations) section of the form:

```
Dim myarray() As String
Dim p As Integer
Dim counter As Integer
```

### In the Form\_Load event:

```
' The last dimension of an array can be re-dimmed
' while preserving data. Therefore, that demension
' should represent "rows" in the grid

ReDim myarray(0 To 2, 0 To 3000) As String
counter = -1

For i = 0 To 3000    'fill the array
    myarray(0, i) = "Column 0 in Row " & i
    myarray(1, i) = "Column 1 in Row " & i
    myarray(2, i) = "Column 2 in Row " & i
    counter = counter + 1
Next i

' Set rows to keep scrollbars accurate
SSDBGrid1.Rows = 3000

For i = 0 To 2
    SSDBGrid1.Columns.Add i
Next i
```

### In the SSDBGrid1\_UnboundAddData event:

```
Dim i As Integer
Dim s As Integer

counter = counter + 1

s = UBound(myarray, 2) 'the present size of the "rows" dimension of
the array
' Increase the size of the array by one
ReDim Preserve myarray(0 To 2, 0 To s + 1)

For i = 0 To 2
    ' Fill in the new row in the array
    myarray(i, s + 1) = SSDBGrid1.Columns(i).Text
Next i
```



## UnboundDeleteRow Event Example

The following example shows how you can use the UnboundDeleteRow event to take a row that has been deleted from the grid and remove it from the array in memory. Also see [Data Grid: Exercise 2](#) for an example of how to use unbound mode with custom code to access a database.

First, place a DataGrid on a form in Visual Basic, and set it to Unbound mode. Then enter the following code:

### In the (declarations) section of the form:

```
Dim myarray() As String
Dim p As Integer
Dim counter As Integer
```

### In the Form\_Load event:

```
' The last dimension of an array can be re-dimmed
' while preserving data. Therefore, that demension
' should represent "rows" in the grid

ReDim myarray(0 To 2, 0 To 3000) As String
counter = -1

For i = 0 To 3000    'fill the array
    myarray(0, i) = "Column 0 in Row " & i
    myarray(1, i) = "Column 1 in Row " & i
    myarray(2, i) = "Column 2 in Row " & i
    counter = counter + 1
Next i

' Set rows to keep scrollbars accurate
SSDBGrid1.Rows = 3000

For i = 0 To 2
    SSDBGrid1.Columns.Add i
Next i
```

### In the UnboundDeleteRow event:

```
Private Sub SSDBGrid1_UnboundDeleteRow(bookmark As Variant)

Dim iSelectedRow As Integer
Dim clone As Variant

    clone = myarray()

For iSelectedRow = SSDBGrid1.SelBookmarks.Count - 1 To 0 Step -1
    ' Mark the selected rows as deleted
    clone(0, SSDBGrid1.SelBookmarks(iSelectedRow)) = "ssdeleted"
Next iSelectedRow

Dim iTempIndex As Integer
' Re-populate the array, but exclude the
' data that has been marked as deleted
```

```

For i = 0 To UBound(myarray, 2)
  If clone(0, i) <> "ssdeleted" Then
    For j = 0 To 2
      myarray(j, iTempIndex) = clone(j, i)
    Next j
    iTempIndex = iTempIndex + 1
  End If
Next i

' This global counter is used by the grids read events, so be sure to
reset it
counter = counter - SSDBGrid1.SelBookmarks.Count
' Shrink the array
ReDim Preserve myarray(2, counter)

SSDBGrid1.SelBookmarks.RemoveAll

' Note that the "Rows" property of the grid can be set in the
AfterDelete event
' to preserve the accuracy of the scrollbars

End Sub

```

### **In the AfterDelete event (optional)**

```

Private Sub SSDBGrid1_AfterDelete(RtnDispErrMsg As Integer)

  SSDBGrid1.Rows = counter
  SSDBGrid1.Refresh

End Sub

```

## UnboundPositionData Event example

The following example shows how you can use the UnboundPositionData event to populate the grid from an array that is stored in memory. This example selects the data to display from the array based on the position of the grid.

First, place a DataGrid on a form in Visual Basic, and set it to Unbound mode. Then enter the following code:

### In the (declarations) section of the form:

```
Dim myarray() As String
Dim p As Integer
Dim counter As Integer
```

### In the Form\_Load event:

```
' The last dimension of an array can be re-dimmed
' while preserving data. Therefore, that demension
' should represent "rows" in the grid

ReDim myarray(0 To 2, 0 To 3000) As String
counter = -1

For i = 0 To 3000    'fill the array
    myarray(0, i) = "Column 0 in Row " & i
    myarray(1, i) = "Column 1 in Row " & i
    myarray(2, i) = "Column 2 in Row " & i
    counter = counter + 1
Next i

' Set rows to keep scrollbars accurate
SSDBGrid1.Rows = 3000

For i = 0 To 2
    SSDBGrid1.Columns.Add i
Next i
```

### In the SSDBGrid1\_UnboundPositionData event:

```
Private Sub SSDBGrid1_UnboundPositionData(StartLocation As Variant,
ByVal NumberOfRowsToMove As Long, NewLocation As Variant)

    If IsNull(StartLocation) Then    'If at beginning or end of data set
then
        StartLocation = 0            'Start at the beginning
    End If

    NewLocation = CLng(StartLocation) + NumberOfRowsToMove

End Sub
```





## UnboundReadData Event Example

The following example shows how you can use the UnboundReadData event to populate the grid from an array that is stored in memory. Also see [Data Grid: Exercise 2](#) for an example of how to use unbound mode with custom code to access a database.

First, place a DataGrid on a form in Visual Basic, and set it to Unbound mode. Then enter the following code:

### In the (declarations) section of the form:

```
Dim myarray() As String
Dim p As Integer
Dim counter As Integer
```

### In the Form\_Load event:

```
' The last dimension of an array can be re-dimmed
' while preserving data. Therefore, that demension
' should represent "rows" in the grid

ReDim myarray(0 To 2, 0 To 3000) As String
counter = -1

For i = 0 To 3000    'fill the array
    myarray(0, i) = "Column 0 in Row " & i
    myarray(1, i) = "Column 1 in Row " & i
    myarray(2, i) = "Column 2 in Row " & i
    counter = counter + 1
Next i

' Set rows to keep scrollbars accurate
SSDBGrid1.Rows = 3000

For i = 0 To 2
    SSDBGrid1.Columns.Add i
Next i
```

### In the SSDBGrid1\_UnboundReadData event:

```
Dim r As Integer
Dim i As Integer
Dim j As Integer

If IsNull(StartLocation) Then    'If the grid is empty then
    If ReadPriorRows Then        'If moving backwards through grid
    then
        p = counter                'pointer = # of last grid row
    Else                            'else
        p = 0                        'pointer = # of first grid row
    End If
Else                                'If the grid already has data in it
then
    p = StartLocation                'pointer = location just before or
    after the row where data will be added
```

```

        If ReadPriorRows Then                'If moving backwards through grid
then
            p = p - 1                        'move pointer back one row
        Else                                  'else
            p = p + 1                        'move pointer ahead one row
        End If
    End If

    'The pointer (p) now points to the row of the grid where you will start
    adding data.

    For i = 0 To RowBuf.RowCount - 1          'For each row in the row
buffer
        If p < 0 Or p > counter Then Exit For 'If the pointer is
outside the grid then stop this

            For j = 0 To 2                    'For each column in
the grid
                RowBuf.Value(i, j) = myarray(j, p) 'Set the value of
each column in the row buffer to the corresponding value in the array
            Next j

            RowBuf.Bookmark(i) = p           'set the value of the
bookmark for the current row in the rowbuffer

            If ReadPriorRows Then            'move the pointer
forward or backward, depending _
                p = p - 1                    'on which way it's
supposed to move
            Else
                p = p + 1
            End If

            r = r + 1                        'increment the number
of rows read
        Next i

        RowBuf.RowCount = r                 'set the size of the
row buffer to the number of rows read

```

By setting the **RowCount** property of the RowBuffer object, you tell the control when to stop reading data. The RowBuffer normally holds ten rows of data. As long as there is enough data to completely fill the RowBuffer, *r* will be 10. When the control reaches the end of the data set it will read fewer than ten rows, *r* will be less than 10 and the **RowCount** of the RowBuffer will be set to a value less than 10. This signals the control that there is no more data to be read, and the **UnboundReadData** event will not be fired again.



## UnboundWriteData Event Example

The following example shows how you can use the UnboundWriteData event to place information that has been modified in the grid into an array that is stored in memory. Also see [Data Grid: Exercise 2](#) for an example of how to use unbound mode with custom code to access a database.

First, place a DataGrid on a form in Visual Basic, and set it to Unbound mode. Then enter the following code:

### In the (declarations) section of the form:

```
Dim myarray() As String
Dim p As Integer
Dim counter As Integer
```

### In the Form\_Load event:

```
' The last dimension of an array can be re-dimmed
' while preserving data. Therefore, that demension
' should represent "rows" in the grid

ReDim myarray(0 To 2, 0 To 3000) As String
counter = -1

For i = 0 To 3000    'fill the array
    myarray(0, i) = "Column 0 in Row " & i
    myarray(1, i) = "Column 1 in Row " & i
    myarray(2, i) = "Column 2 in Row " & i
    counter = counter + 1
Next i

' Set rows to keep scrollbars accurate
SSDBGrid1.Rows = 3000

For i = 0 To 2
    SSDBGrid1.Columns.Add i
Next i
```

### In the SSDBGrid1\_UnboundWriteData event:

```
Private Sub SSDBGrid1_UnboundWriteData(ByVal RowBuf As ssRowBuffer,
WriteLocation As Variant)
    For i = 0 To 2 'Loop through the columns
        myarray(i, WriteLocation) = SSDBGrid1.Columns(i).Value
    Next i
```



## Underline Property example

The following sample code displays the caption text with underline:

```
SSDBOptSet1.Font.Underline = True  
SSDBOptSet1.Caption = "Caption"  
  
SSDBGrid.HeadFont.Underline = True
```

## Value Property (Bookmark Object) example

This sample stores a value to a bookmark:

```
Dim vbookmark as variant
vBookmark = SSDBCommand1.SavedBookmark
SSDBData1.Bookmarks(0).Value = vBookmark
```

## WhereIs Method example

The following example demonstrates the use of **WhereIs**. This code would be placed in the **MouseDown** event sub-routine:

```
Dim nPos As Integer
nPos = SSDBOptSet1.WhereIs(X, Y, Z)
Select Case nPos
    Case 0
        Debug.Print "Pointing at nothing"
    Case 1
        Debug.Print "Pointing at button"
    Case 2
        Debug.Print "Pointing at caption"
EndSelect
```



## **hWndEdit Property**

Applies To

### **Description**

Returns a handle to the edit portion of the grid.

### **Syntax**

*object* . **HwndEdit**

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.

### **Remarks**

This handle is null until the first time the edit is actually used for in-place editing.

## ssRowBuffer Object

[See Also](#)

[Applies To](#)

### Description

Used to transfer data to and from the control in unbound mode.

### Properties

---

<a href="#">Bookmark</a>	ColumnName	<a href="#">RowCount</a>
ColumnCount	<a href="#">ReadType</a>	Value

### Methods

---

### Remarks

The **ssRowBuffer** object is used as a transport mechanism for the unbound grid to send and receive data to/from data storage. The logic for storing and retrieving the data is provided by the programmer as Visual Basic code.

The **ColumnCount** property returns the number of columns present in the object.

The **ColumnName** property returns the name of the specified column in the object.

The **RowCount** property returns or sets the number of rows contained in the **ssRowBuffer** object.

When passed to the **UnboundReadData** event, the **ssRowBuffer** object is used to fill the cache with the rows contained in the object. There may be one or more rows of data in this case.

When passed to the **UnboundWriteData** event, the **ssRowBuffer** object is used to pass the data to the data storage medium. The **UnboundWriteData** event only updates one row at a time, so the **ssRowBuffer** object will only contain one row.

When passed to the **UnboundAddData** event, the data being added is placed in the **ssRowBuffer** object. The **UnboundAddData** event only adds one row at a time, so the **ssRowBuffer** object will only contain one row.

In addition to the values for each column, each row in the row buffer contains a bookmark value. This bookmark value can be set or retrieved to implement indexing of data and perform searches. Bookmark values should be unique if possible.

**Note** If the **ssRowBuffer** is passed in an invalid parameter, it will return an error. See [Error Messages](#) for a list of the codes for these errors and what they mean.

## **ssRowBuffer Object Applies To**

SSDBCombo

SSDBDropDown

SSDBGrid

## **ssRowBuffer Object See Also**

[ReadType](#) property

[UnboundAddData](#) event

[UnboundDeleteRow](#) event

[UnboundReadData](#) event

[UnboundWriteData](#) event

[Performance Tuning](#)

