Data Widgets 3.12

User's Manual

Sheridan

Introducing Data Widgets

Introduction to OCX controls

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.

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.

What's New?

Several important new features have been added to this version of Data Widgets. Data Widgets is now a more complete data management solution, providing a range of advanced features you can use to expand your applications. Most of the improvements are designed to make Data Widgets a more effective tool for presenting data, whether in your application, on paper, or on the Internet or corporate intranet. With Data Widgets 3 you can

now put your data exactly where you want it.

The feature below has been added to Data Widgets 3.1:

- **ADO and OLE DB Support**

The following features have been added to Data Widgets 3.0:

- **Printing**
- **Exporting Data**
- **Save & Restore Grid Layouts**
- **Data Masking**

For information on converting your existing projects to the new version of Data Widgets, see Upgrading to Data Widgets 3.

ADO and OLE DB Support

OLE DB is a specification which was designed to provide an open standard for accessing different types of data. Whereas the Data Access Objects (DAO) and Open Database Connectivity (ODBC) systems found in Visual Basic versions 3, 4 and 5 were created to access relational databases, OLE DB is designed for both relational and nonrelational information sources, such as text and graphical data for the World Wide Web, directory services, and IMS and VSAM data stored in a mainframe.

ADO (ActiveX Data Objects) wraps the complexity of OLE DB into an easy to use object model. You can use ADO to write applications that access and manipulate data using the latest OLE DB providers. ADO's primary benefits are high speed, ease of use, and low overhead.

Data Widgets provides the capability to directly bind to an OLE DB or ADO provider, while still maintaining the ability to bind to a DAO/ODBC data source. You must determine in advanced which type of data binding your application will use. You would then use the correct version(s) of the Data Widgets .OCX files that correspond to your data binding method:

New Data Widgets Control File Names

To support DAO/ODBC data binding, use:	**SSDW3A32.OCX** DataOptionSet, DataCommand, Enhanced Data Control	**SSDW3B32.OCX** DataGrid, DataCombo, DataDropDown
To support ADO/OLE DB data binding, use:	**SSDW3AO.OCX** DataOptionSet, DataCommand, Enhanced Data Control	**SSDW3BO.OCX** DataGrid, DataCombo, DataDropDown

If you wish to bind to an DAO/ODBC data provider (or use any of the other data binding methods provided by Visual Basic version 5 or lower) simply add the standard versions of the Data Widgets 3.1 controls to your project, and continue using the **DataSource**, **DataSourceList**, **DataField**, and **DataFieldList** properties as you have done previously--there is nothing new to learn. When binding to an OLE DB provider, you should add the OLE DB version of the DataWidgets 3.1 controls to your project. You will continue to use the standard data binding properties, but you will also have access to the new OLE DB properties: **DataMember** and **DataMemberList**.

When using an OLE DB data source with the appropriate Data Widgets 3.1 controls, the **DataSource** property is used to identify an available *data environment* which may contain one or more *data members*. A data member (which may also be referred to as a *command*) is similar to a recordset in DAO or ODBC - it represents a group of available data records selected from a data source such as an SQL table or query. Using DAO/ODBC, one recordset provides a single data source, so the **DataSource** property is sufficient to establish the connection. With OLE DB, a data environment may provide multiple data sources, so the additional **DataMember** property is necessary to specify the set of records you wish to use.

When using the DataCombo control, the list portion of the combo is populated from the data environment specified by the **DataSourceList** property, using data from the data member specified by the **DataMemberList** property.

Click a property listed below for specific information regarding how that property is used.

New OLEDB and ADO Related Features

Properties:	DataSource	DataMember
	DataSourceList	DataMemberList

Printing

Data Widgets 3 provides built in report printing capabilities. Printed reports are based on the data displayed in the grid, so all your existing data filtering and layout code can be applied to printed reports. Report layout is based on the grid layout, making it easy for end users to quickly set up data reports via drag and drop. You can choose whether to include grid elements such as column and group headers in the report, or simply print the data. You also have control over how data breaks across pages.

A new object, the **ssPrintInfo** object, gives you control over print-specific properties such as header and footer text, number of copies, page layout, margins and so on. You can specify print parameters through code or display standard Print and Print Setup dialogs to the user and let them configure the print job. You can examine or change the print settings the user has specified before initiating the print job.

As the data is printed, you can monitor the print job on a row-by-row basis, and choose to modify any data before it is printed or even interrupt printing at any point in response to criteria you set.

New Print Related Features

Objects:	ssPrintInfo	
Properties:	Collate	PageHeader
	Copies	PageHeaderFont
	ClippingOverride	PageStart
	DriverOverride	Portrait
	MarginBottom	PrintColors
	MarginLeft	PrintColumnHeaders
	MarginRight	PrinterDeviceName
	MarginTop	PrinterDriverVer
	MaxLinesPerRow	PrintGridLines
	PageBreakOnGroups	PrintGroupHeaders
	PageEnd	PrintHeaders
	PageFooter	RowAutoSize
	PageFooterFont	
Methods:	PrintData	
Events:	PrintBegin	PrintInitialize
	PrintError	RowPrint

Exporting Data

Data Widgets 3 features new data exporting capabilities. You can choose to export the data in your grid to a plain text file or to an Internet-ready HTML file. As with printing, data export is based upon grid contents - you use the SSDBGrid to set up the selection, layout and grouping of the data, then export it to an external file. You can choose whether to include information such as field names, column headers and group headers in the output file.

You can use Data Widgets 3's data exporting abilities to share data from your program with other applications that can read a delimited text file. But the real power of this feature comes from the extensive HTML export capabilities available to you. You can export data into an HTML table that retains much of the formatting and appearance of your DataGrid, essentially creating a World-Wide Web front end for your data. In addition to a block export, you can generate individual HTML files for each row in your grid. This makes it easy to set up master/detail breakdowns of your data.

But the most powerful feature of Data Widgets HTML exporting is the ability to use HTML templates. By specifying a template to use with the exported data, you can create any type of web page you want, and simply insert data from the data source or other grid information (such as column or group names) anywhere you want. Setting up HTML templates is simple, but it gives you unmatched power and flexibility in designing web-ready front ends for your application.

See the HTML Template Codes section for more information on creating and using HTML export templates.

New Export Related Features

Method:	Export
Event:	RowExport

Save and Restore Grid Layouts & StyleSets

Data Widgets 3 now includes the ability to save a customized grid layout in a file and restore it at a later time. You can also choose to save only the StyleSets that have been defined for a grid. You can use this feature to easily provide the user of your program with a way to save the custom grid layout they have created by sizing and moving columns and groups.

You can also create a custom group of StyleSets and use them across multiple controls in one or more projects, providing a consistent look for your applications. You can create a gallery of layouts or views of your data and apply them with a single command. Another option is to create customized grid layouts that change depending on the state of the application or the identity of the user.

Layouts can be saved and restored either at run-time through code or at design-time using the Grid Editor. Use the "Load..." and "Save..." buttons for this purpose.

Note Grid layout files are given a default extension of .GRD by the Grid Editor. At run time, you may use the **LoadLayout** and **SaveLayout** methods to work with files that have any extension, but to use your layout files with the Grid Editor at design-time, you should give them a .GRD extension. The Gride Editor will not recognize as valid any layout files with extensions other than .GRD.

New Layout Save & Restore Features

Methods:	LoadLayout	SaveLayout

Data Masking

You can now mask input of any column in the data grid or the data combo. By specifying a mask string for a DataGrid column or DataCombo, you can restrict the type of data the user can enter, and perform validation checking on the type of data entered. The data masking features of Data Widgets 3 are comparable to the data masking provided by the Microsoft Masked Edit Control, which is included with the Professional & Enterprise editions of Visual Basic.

New Data Masking Related Features

Properties:	ClipMode	PromptChar
	Mask	PromptInclude
Event:	ValidationError	

Object Concepts

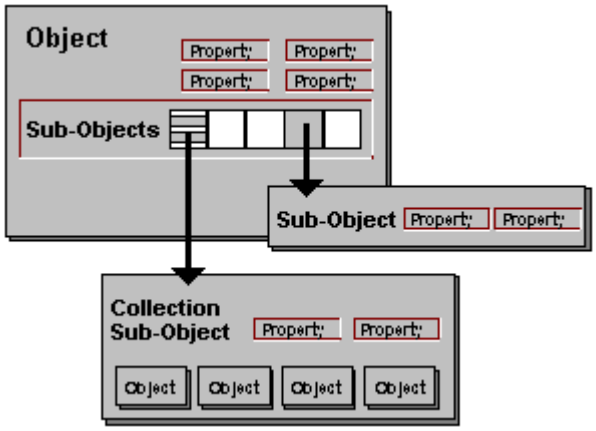
This section will be of special interest to programmers who have worked with earlier versions of our custom controls. It highlights the major differences between the older controls you may be familiar with and the newer controls you now have.

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

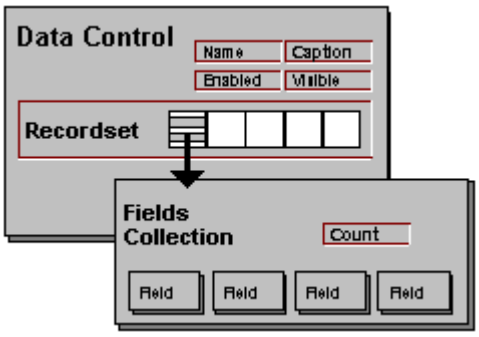
controls.

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 With.. End With statements:

```
With SSDBGrid1.Columns(4)
   .Alignment = 0
   .BackColor = vbRed
   .ForeColor = vbWhite
   .Caption = "Column 5"
EndWith
```

Data Grid Control

Data Grid Overview

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.

Data Grid Features

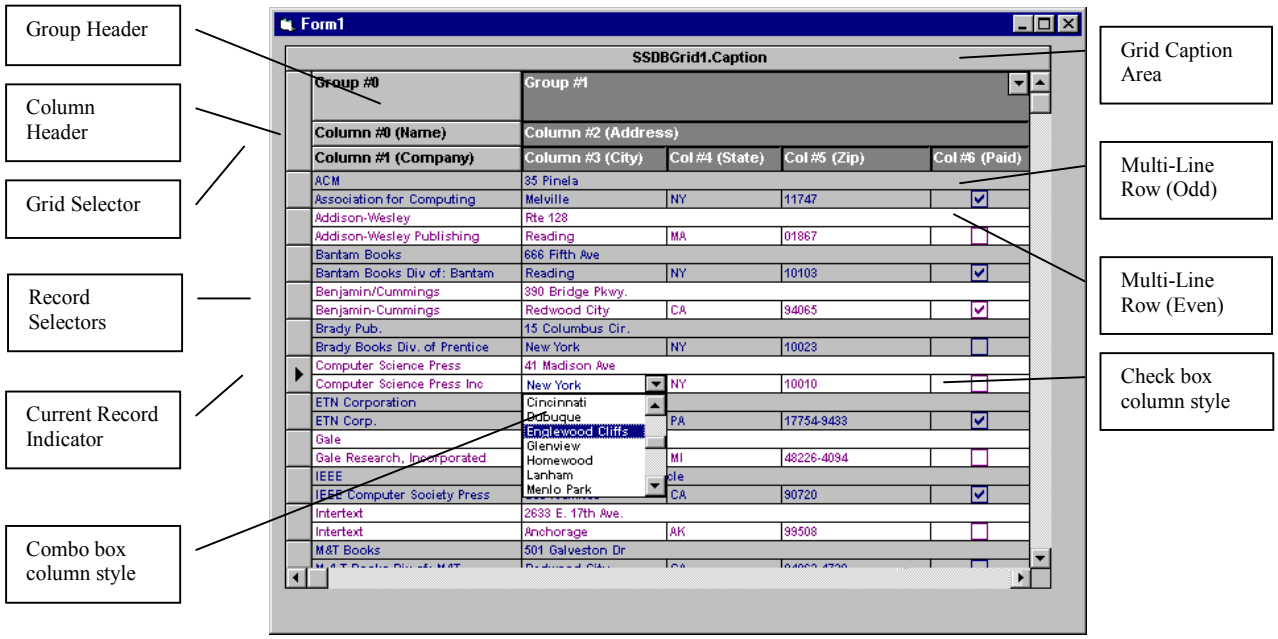
The Data Grid is a fully editable bound grid that allows you to edit an entire record set, regardless of size, on screen without writing any code.

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
- Mask and validate any data input by the user
- Save grid layout and StyleSets to a file
- Load grid layout and StyleSets from a file
- Print out reports of grid data
- Export grid data to a text file
- Export grid data to an HTML file or a series of HTML files

Anatomy of a Data Grid



Data Combo Control

Data Combo Overview

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.

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
- Save grid layout and StyleSets to a file
- Load grid layout and StyleSets from a file
- Mask and validate any data input by the user

Anatomy of a Data Combo

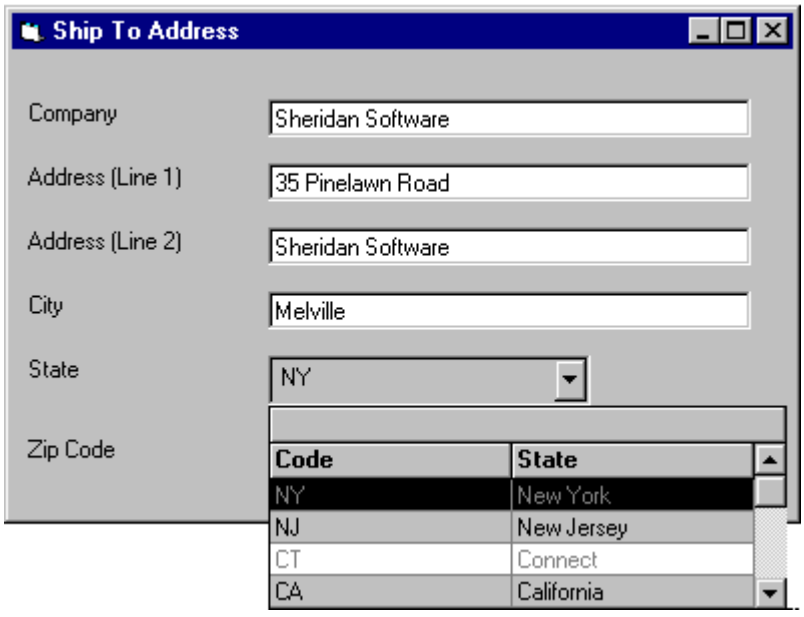
Although there are major fundamental differences, the Data Combo control looks and behaves much like a standard Windows combo box. The major difference is that the Data Combo can be bound to a data control.

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 this:



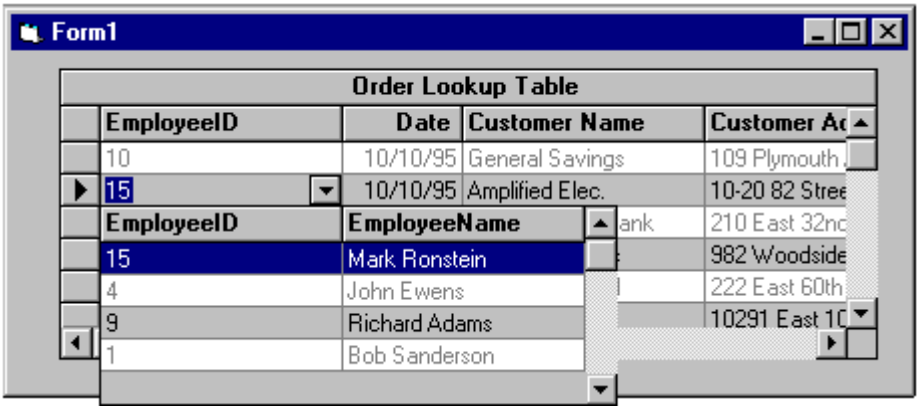
Data DropDown Control

Data DropDown Overview

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, which must be 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.



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
- Save grid layout and StyleSets to a file
- Load grid layout and StyleSets from a file
- Full design time capabilities

Enhanced Data Control

Enhanced Data Control Overview

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.

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.

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.

Data OptionSet Control

Data OptionSet Overview

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.

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

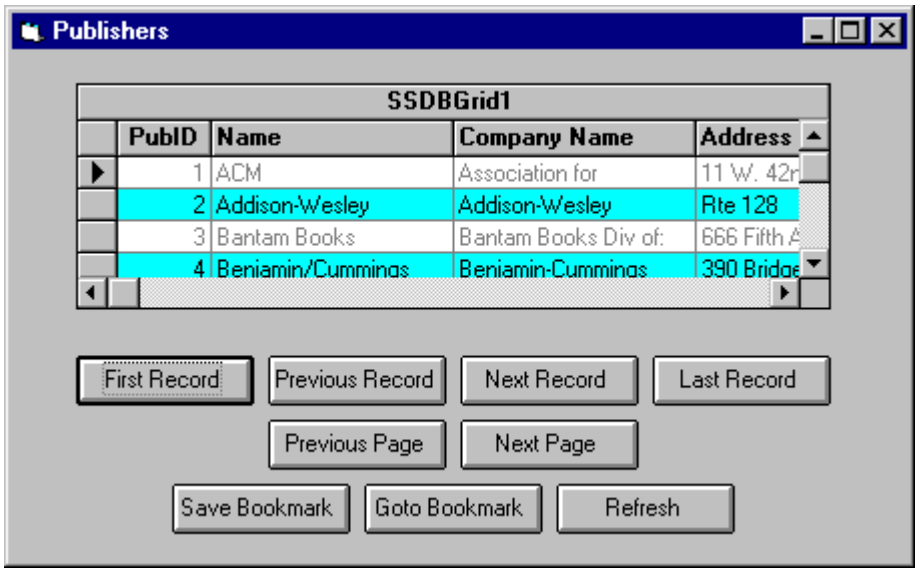
Data Command Button Control

Data Command Button Overview

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



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

Learning About Data Widgets

Data Grid Guided Tour

DBGrid: 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.

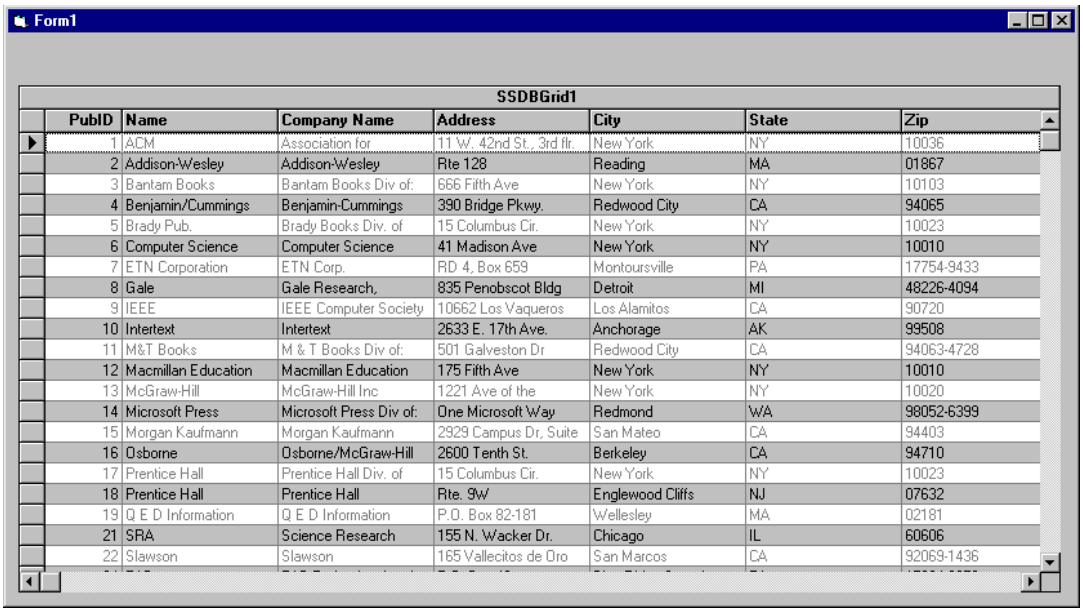
In this exercise, you will create an application that makes use of the Data Grid control. The database used will be the BIBLIO.MDB that ships with Visual Basic and is located in the Visual Basic home directory. If you are using an environment other than Visual Basic, and do not have the BIBLIO.MDB database,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

DBGrid Ex. 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:



PubID	Name	Company Name	Address	City	State	Zip
1	ACM	Association for	11 W. 42nd St., 3rd flr.	New York	NY	10036
2	Addison-Wesley	Addison-Wesley	Rte 128	Reading	MA	01867
3	Bantam Books	Bantam Books Div of:	666 Fifth Ave	New York	NY	10103
4	Benjamin/Cummings	Benjamin-Cummings	390 Bridge Pkwy.	Redwood City	CA	94065
5	Brady Pub.	Brady Books Div. of	15 Columbus Cir.	New York	NY	10023
6	Computer Science	Computer Science	41 Madison Ave	New York	NY	10010
7	ETN Corporation	ETN Corp.	RD 4, Box 659	Montoursville	PA	17754-9433
8	Gale	Gale Research,	835 Penobscot Bldg	Detroit	MI	48226-4094
9	IEEE	IEEE Computer Society	10662 Los Vaqueros	Los Alamitos	CA	90720
10	Intertext	Intertext	2633 E. 17th Ave.	Anchorage	AK	99508
11	M&T Books	M & T Books Div of:	501 Galveston Dr	Redwood City	CA	94063-4728
12	Macmillan Education	Macmillan Education	175 Fifth Ave	New York	NY	10010
13	McGraw-Hill	McGraw-Hill Inc	1221 Ave of the	New York	NY	10020
14	Microsoft Press	Microsoft Press Div of:	One Microsoft Way	Redmond	WA	98052-6399
15	Morgan Kaufmann	Morgan Kaufmann	2929 Campus Dr, Suite	San Mateo	CA	94403
16	Osborne	Osborne/McGraw-Hill	2600 Tenth St.	Berkeley	CA	94710
17	Prentice Hall	Prentice Hall Div. of	15 Columbus Cir.	New York	NY	10023
18	Prentice Hall	Prentice Hall	Rte. 9W	Englewood Cliffs	NJ	07632
19	Q E D Information	Q E D Information	P.O. Box 82-181	Wellesley	MA	02181
21	SRA	Science Research	155 N. Wacker Dr.	Chicago	IL	60606
22	Slawson	Slawson	165 Vallecitos de Oro	San Marcos	CA	92069-1436

DBGrid Ex. 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.

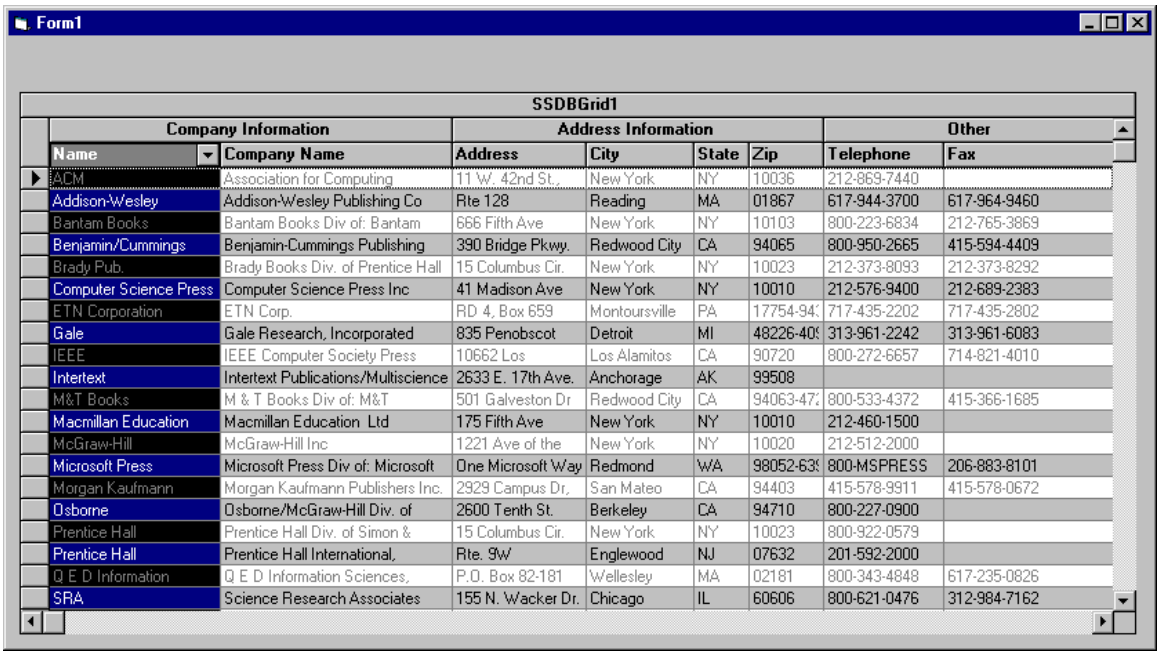
1. Select the "Columns" tab. The Columns tab allows us to create and delete column definitions.
2. Click on the group header labeled "Company Information".
3. Click the **Fields** button. The Fields button allows us to automatically create columns from a bound database.
4. Select the fields "Name" and "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 three 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:



DBGrid Ex. 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 the section covering the Data Grid control.

Set the following properties as shown:

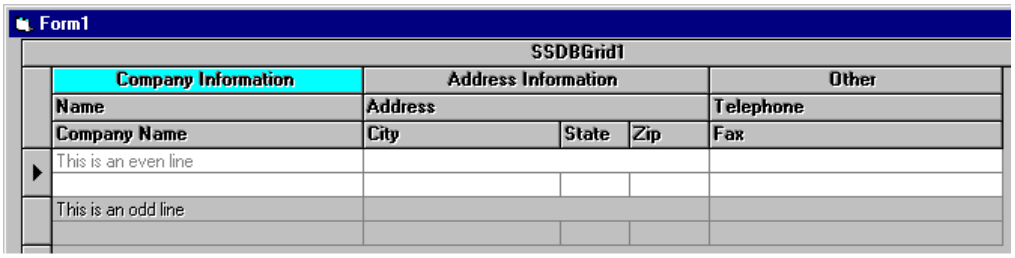
Property	Value	What It Does
Caption	"Publisher Database"	Specifies the caption title for the Data Grid
CaptionAlignment	"0 - Left Justify"	Left justifies the caption
AllowAddNew	True	Allows users to add new records to the Data Grid
AllowDelete	True	Allows users to delete records from the Data Grid
Font	"Arial" for Name 8 for Point Size	Specifies the font name and size to be used for the grid text.
HeadFont3D	"Inset w/light shading"	Gives the text of your grid headers a 3D appearance.
HeadFont	"Arial" for Name 12 for Point Size	Specifies the font name and size to be used for the grid

		headers, including caption.
AllowColumnMoving	"2 - Anywhere"	Allows users to move columns anywhere on the grid
AllowColumnSwapping	"2 - Anywhere"	Allows users to swap columns anywhere on the grid
DividerType	"Horizontal"	Determines what type of row divider is used.
SelectByCell	True	Allows the selection of an entire row if the user clicks on a cell.
GroupHeadLines	2	Allows the Group Headers to occupy two rows.
LevelCount	2	Allows each record to occupy two rows. You will need to decide what columns you want on each level.

Customizing Group options

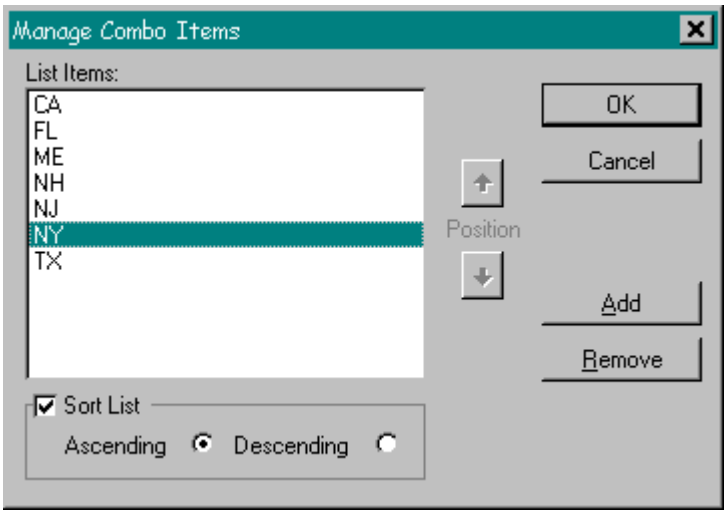
1. Select the "Groups" tab.
2. Select the group "Other" from the **Name** list.
3. Set the **AllowSizing** property to False. This prevents users from changing the width of the "Other" group.
4. Select the group "Company Information" from the **Name** list.
5. Set the **HeadBackColor** property to the 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:



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.

DBGrid Ex. 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.

DBGrid: Exercise 2 (Unbound Mode) - Part I

In this exercise, you will create a fully functional unbound Data Grid control. The database used will be the `BIBLIO.MDB` that resides in the `\SAMPLES\CHAP07` sub-directory of the Data Widgets installation directory.

This sample uses a Recordset object to handle the transfer of data between the Grid and the database. A Recordset object behaves much like the Visual Basic Data control, except that it is declared through code and does not require a control to be placed on your form.

Although this may seem similar to the way the Bound Grid operates, the Unbound Grid actually offers much more flexibility. Once you learn how to manage Grid data in unbound mode, you can easily adapt the DataGrid (or

DataCombo or DataDropDown) to handle data from any object-based data source, such as Remote Data Objects or any data provider that you or a third party created.

In Unbound mode, you communicate with the Grid by accessing an object called the *Row Buffer* (see ssRowBuffer Object for more details). The Row Buffer is a temporary storage area that both contains data that is being manipulated and communicates with your program about how much and what kind of data is needed. You examine and set the properties of the ssRowBuffer object to control the flow of data into and out of the Grid.

In the following steps, you will see how to use the Row Buffer to populate the Grid and handle ordinary data chores, such as adding and deleting records or editing existing data.

Note This exercise makes use of the Microsoft DAO 2.5/3.0 Object Library by using commands such as OpenDatabase and OpenRecordset. If you do not have the correct reference to this library in your project, Visual Basic 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 *Project* menu.

1. Place an 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. Leave some space between the bottom of the Grid and the bottom of the form.
2. Set the **DataMode** property of the DataGrid to '1 - Unbound'. This places the Grid in unbound mode. In Unbound mode, whenever the Grid needs to display, change, add or delete data, it will pass control to your application.

   When the Grid is in Unbound mode, it fires one or more of the Unbound events (**UnboundAddData**, **UnboundDeleteRow**, **UnboundPositionData**, **UnboundReadData** & **UnboundWriteData**) whenever it needs to perform some data operation. These are the events you use to create your custom data handling code. You must write Visual Basic code in these events to handle the transport of data into and out of the Grid.
3. Set the **AllowAddNew** and **AllowDelete** properties of the DataGrid to 'True'. This will make sure that all of the Unbound events will be fired, including the ones that handle the addition and deletion of data. These settings also allow the user of your application to add and delete records via the Grid.
4. Add the following code to the **(General)(declarations)** section of your form:

```
Dim dbData As Database
Dim rsRSet As Recordset
Dim iFldCount As Integer
```

   These statements set up the global variables that will be used throughout the program. `dbData` and `rsRSet` are object variables that will contain the Database and Recordset objects used to provide the data. `iFldCount` will be used as a field counter.
5. Add the following code to the **Form_Load** procedure:

```
Dim iFld As Integer
   Set dbData = OpenDatabase("biblio.mdb")
   Set rsRSet = dbData.OpenRecordset("Titles")
   iFldCount = rsRSet.Fields.Count
```

   This first block of code creates the data access objects and determines the number of fields in the recordset. The number of fields will determine the number of columns that will be added to the Grid. Because these are form-level variables, they will be accessible in any event procedure.

   Note You may need to change the path in the OpenDatabase statement to point to the directory that contains your copy of `BIBLIO.MDB`. A copy of this file is included with Data Widgets and can be found in the `SAMPLES\CHAP07` subdirectory under the directory where Data Widgets is installed.
6. Add the following code to the same event, immediately following the code you just entered:

```
SSDBGrid1.Columns.RemoveAll

For iFld = 0 To (iFldCount - 1)
   SSDBGrid1.Columns.Add iFld
   SSDBGrid1.Columns(iFld).Caption = rsRSet.Fields(iFld).Name
Next iFld
```

This code removes any existing columns from the Grid. (There shouldn't be any at this point, but if columns had been added to the Grid at design time, this step would be necessary. It is included here as a precaution.) It then adds one column to the Grid for every field in the recordset, as determined by the value of `iFldCount`. It also sets the caption of each new column to the name of the field the column will display.

7. Add the following code to the **UnboundReadData** event of the DataGrid:

```
Dim iRBRow As Integer
Dim iGridRows As Integer
Dim iFld As Integer

   iGridRows = 0
```

This code initializes the procedure by declaring the variables that will be used to move data into the Grid. `iRBRow` will be used to count the number of rows of data requested by the ssRowBuffer object (`RowBuf`) that is passed to the event. `iGridRows` will count how many rows of data should be supplied to the Grid from the data source. `iFld` will be used as a generic counter when pulling data from the recordset. Setting iGridRows to 0 indicates that, at the start of the event, no rows have been read from the recordset.

Continue the code of this event by adding the following:

```
If IsNull(StartLocation) Then 'If the Grid is empty
   If ReadPriorRows Then      'If ReadPriorRows is True
      rsRSet.MoveLast         'then the Grid is being
   Else                       'scrolled up towards the top
      rsRSet.MoveFirst
   End If
Else                          'If Grid contains data
   rsRSet.Bookmark = StartLocation
   If ReadPriorRows Then
      rsRSet.MovePrevious
   Else
      rsRSet.MoveNext
   End If
End If
```

This code takes two different actions depending on whether the Grid is empty or already contains data. The *StartLocation* parameter will only be Null when there is no data in the Grid. Otherwise, it will contain the bookmark of the row that immediately precedes the data that is about to be displayed. (If the Grid is being scrolled upwards, it will be the bookmark of the row that immediately follows the data to be displayed.)

If *StartLocation* is null and the Grid is empty, the next step is to determine which direction the Grid is being scrolled. Although the most likely scenario is that the user is starting at the top of the recordset and scrolling down, it is also possible that the Grid could be populated from the end of the recordset, with the user scrolling up. This code accommodates both scenarios.

The *ReadPriorRows* parameter is the key to determining which direction the Grid is being scrolled. If the Grid is scrolling down, *ReadPriorRows* is set to False. For upwards scrolling, *ReadPriorRows* becomes True. Based on the value of *ReadPriorRows*, the recordset is moved to either the end or the beginning in preparation for populating the empty Grid.

If the Grid is not empty, the recordset must be moved to the location where the data read will begin. Since *StartLocation* is a bookmark value, you can simply assign the value of *StartLocation* to the Bookmark property of the recordset and the repositioning of the recordset will take place automatically.

However, since *StartLocation* represents the bookmark of a record that is already displayed by the Grid, the recordset must be positioned to the *following* record before the data read can begin. (This is true only if the Grid is being scrolled down. As before, if the Grid is being scrolled up, the recordset must be positioned to the *preceding* record.) The last block of code uses *ReadPriorRows* to determine the direction of the scrolling and then moves the recordset forward one record or back one record, whichever is appropriate. Once the recordset is properly positioned, you can begin transferring data into the Grid.

8. Add the following code to the event procedure:

```
For iRBRow = 0 To RowBuf.RowCount - 1
```

```
If rsRSet.BOF Or rsRSet.EOF Then Exit For
```

This code initiates the loop that will fill the ssRowBuffer object with data to display in the Grid. It then immediately tests the recordset to see if there is any data to be read. If there isn't (because the beginning or end of the recordset has been reached) it exits the loop.

The ssRowBuffer is used to shuttle data into and out of the Grid. It is also the mechanism you use to notify the Grid that it has read all the available data. During an unbound data read, the Grid will continually fire the **UnboundReadData** event until one of two things happens:

The Grid is completely populated and no more data is required

You notify the Grid that there is no more data available

In the **UnboundReadData** event, the ssRowBuffer can contain a maximum of ten rows of data. (In other Unbound events, the ssRowBuffer moves only one row of data at a time.) The ssRowBuffer may also contain any number of rows up to ten. The number of rows in the ssRowBuffer (as determined by its `RowCount` property) is an indication of how much data is being requested for display. If you supply the number of rows requested, the data is used to populate the Grid and if more data is needed the **UnboundReadData** event is fired again to retrieve more data.

Each time the event is entered, the ssRowBuffer object is emptied, and its `RowCount` property is set to the number of rows of data needed by the Grid (up to the ten row maximum.) You signal the end of the data read by failing to provide the ssRowBuffer with the number of rows requested. If the **UnboundReadData** event ends, and you have added fewer than `RowBuf.RowCount` rows to the ssRowBuffer, the event will not be fired again until the next time the Grid must be repopulated with data.

9. Add the following code to the event procedure, immediately following the previous code:

```
Select Case RowBuf.ReadType
Case ssReadTypeAllData         'All data must be read
   RowBuf.Bookmark(iRBRow) = rsRSet.Bookmark
   For iFld = 0 To (iFldCount - 1)
      RowBuf.Value(iRBRow, iFld) = rsRSet(iFld)
   Next iFld
Case ssReadTypeBookmarkOnly    'Only bookmarks must be read
   RowBuf.Bookmark(iRBRow) = rsRSet.Bookmark
End Select      'Cases 2 and 3 are not used by DBGrid
```

This is the code that actually populates the ssRowBuffer with data from the recordset. Because it is inside the `For iRBRow...` loop, it will be executed once for each row in the ssRowBuffer. It makes use of the **ReadType** property of the ssRowBuffer object to optimize the data read and improve the efficiency and speed of the Grid.

There are two situations which require that the Grid perform an unbound read. The first is that data must be displayed in response to the user scrolling up or down in the Grid, either via the keyboard (Page Up & Down or the Up & Down Arrow keys) or by clicking on the Grid's scrollbar. In this case, data must be supplied to the Grid so that the display can be updated on an ongoing basis.

The second situation occurs when the user drags the scrollbar thumb up or down and releases it, or uses a keystroke such as CTRL-HOME or CTRL-END to move up or down an arbitrary number of records. In this case, a data read must occur so that the Grid can determine where in the recordset to begin requesting data. When the Grid performs this second type of read, it does not require any data from the record source, it is only looking for position information in the form of bookmarks. Any other data you supply, such as field contents, is discarded. Once the Grid has "oriented" itself with this positional data read, a "regular" data read can be performed to actually display the data, the same as in the first situation.

The **ReadType** property is how the Grid tells you which type of read is being performed. By testing for the value of this property, you can avoid returning data during a positional data read when it is not needed. The code above checks the value of **ReadType** and decides what information to add to the ssRowBuffer. If this is a full data read, the Bookmark of the current RowBuffer row is set to the current Bookmark of the recordset, then the current RowBuffer row is filled with field data from the current row in the recordset. If this is a position-only read, only the Bookmark of the current recordset row is assigned to the current RowBuffer row.

ReadType has other values that apply only to the DBCombo and DBDropDown controls. When using

ReadType with the DBGrid, you do not need to test for values greater than 1.

10. Add the following code to the end of the event procedure:

```
        If ReadPriorRows Then
           rsRSet.MovePrevious
        Else
           rsRSet.MoveNext
        End If

        iGridRows = iGridRows + 1

    Next iRBRow

RowBuf.RowCount = iGridRows
```

This last block of code finishes up the **UnboundReadData** event. Now that one row of the row buffer has been supplied with data (either a Bookmark and field data, or just a Bookmark) the recordset is moved on to the next record (or previous record, depending on the direction the Grid is being scrolled.) The counter tracking the number of rows retrieved is incremented by one, and the loop branches back to populate the next row in the ssRowBuffer.

At some point the loop ends, either because each row in the ssRowBuffer has been dealt with, or because the recordset has run out of data, triggering the `Exit For` statement you entered in step 9. If the loop has added data to each row in the ssRowBuffer, `iGridRows` will be equal to `RowBuf.RowCount`. When the last statement is executed, the value of `RowBuf.RowCount` will not change, and the Grid will fire the **UnboundReadData** event again if it requires more data. If the Grid has already retrieved all the data it needs, it will not fire the event again.

If the loop ends as a result of reaching the beginning or end of the recordset, `iGridRows` will be less than `Rowbuf.RowCount`, and the number of rows in the ssRowBuffer will be changed to reflect this smaller number. When the event ends, the Grid will determine that the ssRowBuffer contains fewer rows than it requested, and the **UnboundReadData** event will not be fired again.

Note that the number of times the event is fired depends partly upon the number of visible rows the Grid must display. For example, if your Grid displays fifteen rows, the **UnboundReadData** event may fire as few as two times. The first time, ten rows are requested by the ssRowBuffer, supplied and moved into the Grid. The second time the event fires, only five rows remain to be populated and the ssRowBuffer will contain only five rows when it is passed to the event. Note that issues other than the number of Grid rows may affect how many rows are requested by the ssRowBuffer, so the number of times the event is called will not always correspond directly to the number of rows in the Grid. In general, your code should never assume either a fixed number of rows in the ssRowBuffer or a fixed number of calls to the **UnboundReadData** event.

Now you have seen how the **UnboundReadData** event works with the ssRowBuffer object to populate the Grid with data. Part two of this exercise covers how the **UnboundPositionData** event works when the user repositions the Grid. Part two also discuses the rest of the unbound events, which handle the addition, editing and deletion of data from the recordset in response to the user working with data in the Grid.

DBGrid: Ex. 2 (Unbound Mode) - Part II

In the first part of this exercise, you saw how the ssRowBuffer is passed to the **UnboundReadData** event to retrieve up to ten rows of data for display in the Grid. You also saw how the **ReadType** property is used to determine what type of data read is occurring. In some instances, the Grid performs a data read only to determine its position within the recordset. By returning just Bookmarks (instead of Bookmarks and field data) during these "positional" data reads, the event code is optimized and the Grid's performance is enhanced.

In this part of the exercise, you will see more of what happens when the unbound Grid is repositioned. You will see how to use the **UnboundPositionData** event to align the Grid and the recordset to the same location using Bookmarks.

1. Double-click the Grid on your form to open the code window (if it is not already open.) Select the **UnboundPositionData** event from the dropdown.
2. Add the following code to the beginning of the event:

```
If IsNull(StartLocation) Then
   'Going up or down?
   If NumberOfRowsToMove = 0 Then
      Exit Sub
   ElseIf NumberOfRowsToMove < 0 Then
      rsRSet.MoveLast
   Else
      rsRSet.MoveFirst
   End If
```

This code tests the *StartLocation* parameter that was passed to the event to see if it is Null. (*StartLocation* is the location of the row directly before or after the row that must be retrieved next.) *StartLocation* may be Null if the rows to be retrieved are near or at the beginning or the end of the data file.

If it is Null, the next block of code determines the direction of the position based on the value of *NumberOfRowsToMove*. This is similar to the way *ReadPriorRows* was used in **UnboundReadData**. The difference is that *ReadPriorRows* is a Boolean value, whereas *NumberOfRowsToMove* is a numeric value that is either positive, negative or zero. If the value of *NumberOfRowsToMove* is positive, the Grid is being scrolled downward. If it is negative, the Grid is scrolling upward. Based on this value, the recordset is moved to either the beginning or the end so it is ready to begin populating the empty Grid..

3. Add the following code to the event, immediately following the previous code:

```
Else
   rsRSet.Bookmark = StartLocation
End If

'Note: Do not use StartLocation here - it could be null
rsRSet.Move NumberOfRowsToMove

NewLocation = rsRSet.Bookmark
```

The first part of this code completes the `If` loop that started the procedure, providing the code that will be executed when the Grid is not empty. The Bookmark property of the recordset is simply set to the value of the *StartLocation* parameter. Since Grid row bookmarks always correspond exactly to their counterparts in the recordset, this ensures the recordset is in the same place as the Grid before the reposition takes place.

The next line of code moves the current position of the recordset a number of records equal to the number of Grid rows that have been scrolled during the reposition. Note that because *NumberOfRowsToMove* can be positive or negative, this line will move the recordset either forwards or backwards. Note that some data connections (ADO, for example) will allow you to specify a starting point as well as the number of records when calling the `Move` method of the recordset. This will produce an error if *StartLocation* is null, so it is important to separate the assignment of the starting point and the movement of the recordset.

The last line of code sets the value of the *NewLocation* parameter to the Bookmark of the newly-repositioned recordset. This determines the value that will be passed as the *StartLocation* parameter of the **UnboundReadData** event. This event will be called immediately after the reposition takes place to fill the Grid with data from the new location.

4. Add the following code to the **UnboundAddData** event:

```
Dim iFld As Integer

   rsRSet.AddNew

   For iFld = 0 To (iFldCount - 1)
      If Not IsNull(RowBuf.Value(0, iFld)) Then
         rsRSet(iFld) = RowBuf.Value(0, iFld)
      End If
   Next iFld

   rsRSet.Update
   rsRSet.MoveLast
```

```
NewRowBookmark = rsRSet.Bookmark
```

When the **UnboundAddData** event occurs, the data that must be added is already in the ssRowBuffer. All you have to do is write code that adds a new record to the recordset and transfers the records from the ssRowBuffer into the newly added record. That is what the first part of this code does. The loop includes a check for Null values because empty fields in the Grrid will be passed to the ssRowBuffer as Null values. You can optimize your code and avoid problems with your data by screening out these values when sending data to the recordset.

The last three lines of code take care of cleaning up after the new record is added. The recordset is updated so that the changes are committed to the database, then the recordset is moved to the new record, which has been added to the end of the recordset. Finally, the bookmark of the new record is assigned to the *NewRowBookmark* parameter of the event to signal that the addition was successfully completed.

5. Add the following code to the **UnboundWriteData** event:

```
Dim iFld As Integer

   rsRSet.Bookmark = WriteLocation
   rsRSet.Edit

   For iFld = 0 To (iFldCount - 1)
      If Not IsNull(RowBuf.Value(0, iFld)) Then
         rsRSet(iFld) = RowBuf.Value(0, iFld)
      End If
   Next iFld

   rsRSet.Update
```

This code operates in a manner similar to the code for the **UnboundAddData** event. The Bookmark of the data record to be written is passed to the event as a parameter and used to position the recordset. The recordset is then placed in edit mode. The ssRowBuffer contains a single row of field data that will be used to replace the existing data. The contents of the ssRowBuffer are transferred into the recordset one field at a time, and then the recordset is updated to commit the changes.

6. Add the following code to the **UnboundDeleteRow** event:

```
rsRSet.Bookmark = Bookmark
rsRSet.Delete
```

This code simply points to the record in the recordset that is being deleted, using the *Bookmark* parameter passed to the event. The delete is then performed.

This concludes the first Unbound Mode tutorial. You have seen in detail how each of the Unbound methods are used to move data into and out of the Grid using the ssRowBuffer object, and how the recordset is positioned according to the parameters passed to each event. By now you should have a good understanding of how the events function together and how the Grid can communicate with external data sources through code.

In the next exercise, you will learn more about Unbound mode by seeing how the Grid can be bound to an array in memory to create a completely virtual data environment.

DBGrid: Exercise 3 (Unbound Mode)

In this exercise, you will continue to explore the use of the Unbound Mode of the DataGrid by connecting the Grid to a memory array. Although the DataGrid is generally used as the front end for data stored in a database management system, this is just one of its functions. The Unbound mode of the Grid is flexible enough to interact with any method of data storage - even ones you may create yourself.

Memory arrays are often used to hold data in memory on a temporary basis, while a program is running. Data stored in the array can be manipulated quickly and easily, and then saved to permanent storage as the need arises.

To begin this exercise, you will need a Visual Basic project with a single form. The Data Widgets controls should be included in the project. If they are not, add them using the Components option of the Project menu.

1. Place an 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 of the DataGrid to '1 - Unbound'. This places the Grid in unbound mode.
4. Set the **AllowAddNew** and **AllowDelete** properties of the DataGrid to 'True'.
5. Add the following code to the **(General) (Declarations)** section of the form:

```
Dim UBArray() As String
Dim vClone As Variant
Dim iNewSize As Integer
Dim iCols as Integer
Dim iPoint As Integer
Dim iCount As Integer
```

   This code declares the array, plus the form-level pointer and counter variables that will be used in the Unbound events.

6. Add the following code to the **Load** event of the form:

```
Dim iCol as Integer
Dim iC as Integer

   iCols = 5
   ReDim UBArray(0 To (iCols-1), 0 To 299) As String
   iCount = -1
```

   The **ReDim** statement can be used to re-dimension an existing array. If the **Preserve** keyword is used, the **ReDim** will not destroy data that is stored in the array. When you re-dimension the array, you will change the second dimension of the array, which corresponds to the rows of the Grid. The first dimension will correspond to the columns of the grid, and will remain the same throughout the application. This array will have five columns.

   The `iCount` variable tracks which "rows" in the array contain data. (`iCount` is equal to the highest "row number" in the grid. Because array subscripts are zero-based, this number effectively equals the number of rows in the array minus 1.) The array is currently empty, so `iCount` is set to -1.

   Enter the following code next:

```
For iC = 0 To UBound(UBArray, 2)
   For iCol = 0 to (iCols - 1)
      UBArray(iCol, iC) = "Row " & iC & " in Column " & iCol
   Next iCol
   iCount = iCount + 1
Next iC
```

   This code fills the array with as many rows of sample data as it can hold. As each row is added, the row counter variable is increased. Now enter the following:

```
SSDBGrid1.Rows = iCount

For iCol = 0 To (iCols - 1)
   SSDBGrid1.Columns.Add iCol
Next iCol
```

   This code sets the **Rows** property of the Grid so that the Grid's scrollbar can accurately reflect the position of the current row relative to the total number of rows. In some cases, the total number of rows in a data set is unknown, or finding out the total number of rows may adversely impact performance. If you were using a recordset, for example, you would probably have to perform a **MoveLast** on the recordset to accurately retrieve the number of rows. However, in this case your program has immediate access to the total number of rows, so you can easily take advantage of this feature.

   The second block of code simply adds enough columns to the Grid to match the number of "columns" (first-dimension elements) in the array.

7. Select the **UnboundReadData** event of the Grid and enter the following code for the event procedure:

```
Dim iRBRow As Integer
Dim iCol As Integer
Dim iRow As Integer

   iRow = 0

   If IsNull(StartLocation) Then
      If ReadPriorRows Then
         iPoint = iCount
      Else
         iPoint = 0
      End If
   Else
      iPoint = StartLocation
      If ReadPriorRows Then
         iPoint = iPoint - 1
      Else
         iPoint = iPoint + 1
      End If
   End If
```

This code may look familiar - it is similar to the code used in step 7 of Exercise Two, Part 1. The only difference is that instead of using commands to move the location within the recordset, this code uses a variable (`iPoint`) as a pointer to a row of data within the array. Overall, the effect is the same. Since `iCount` is equal to the total number of data rows in the array, setting `iPoint` equal to `iCount` is the equivalent to invoking the **MoveLast** method of a recordset. Similarly, setting `iPoint` equal to zero is the equivalent of performing a **MoveFirst**. The use of the *StartLocation* and *ReadPriorRows* parameters is unchanged.

The second block of code simply increments or decrements the pointer value by one if the *StartLocation* is not Null. This is the equivalent action to performing a **MoveNext** or a **MovePrevious** on a recordset.

Next, add the following code to the event procedure:

```
   For iRBRow = 0 To RowBuf.RowCount - 1
      If iPoint < 0 Or iPoint > iCount Then Exit For

      For iCol = 0 To (iCols - 1)
         RowBuf.Value(iRBRow, iCol) = UBArray(iCol, iPoint)
      Next iCol
      RowBuf.Bookmark(iRBRow) = iPoint
      If ReadPriorRows Then
         iPoint = iPoint - 1
      Else
         iPoint = iPoint + 1
      End If
      iRow = iRow + 1
   Next iRBRow

   RowBuf.RowCount = iRow
```

This code loads the ssRowBuffer with data from the array. As in the recordset example, the outer `For` loop iterates through each row in the ssRowBuffer, and ends if it runs out of array data to load. For each ssRowBuffer row, the inner `For` loop reads a column of data from the array. The important thing to notice about this loop is the way the indexes differ between the ssRowBuffer and the array. When assigning data to the `RowBuf.Value` property, the row index is followed by the column index (`iRBRow, iCol`). However, when addressing the array, the column index comes first, followed by the row index as indicated by the pointer variable (`iCol, iPoint`).

Once the ssRowBuffer row is loaded, the ssRowBuffer's Bookmark is set to the value of the pointer variable. Because you are dealing with an array, the Bookmark represents an absolute row number rather than a reference to a database record. Once the data and the Bookmark have been assigned, the pointer variable is updated to point to the next array element to be read, depending on the direction the Grid is being scrolled. The value of `iRow` is incremented to track the actual number of array rows that have been read. Finally, `iRow` is assigned to the RowCount property of the ssRowBuffer to either complete or continue the data read.

You may notice that the optimization code used in the recordset example is not used here. Because all data is stored in memory, the benefits from optimizing the data read are not as significant. However, if you wanted to you could include the optimization code here as well.

8. Select the **UnboundPositionData** event of the DataGrid and enter the following code:

```
If IsNull(StartLocation) Then
   StartLocation = 0
End If

NewLocation = CLng(StartLocation) + NumberOfRowsToMove
```

This code simply adjusts the values of the *StartLocation* and *NewLocation* parameters so that they will be set to the appropriate values when the event ends.

9. Enter the following code in the **UnboundAddData** event:

```
Dim iSize As Integer
Dim iCol As Integer

   iCount = iCount + 1

   iSize = UBound(UBArray, 2)
   ReDim Preserve UBArray(0 To (iCols - 1), 0 To iSize + 1)

   For iCol = 0 To (iCols - 1)
      If Not IsNull(RowBuf.Value(0, iCol)) Then
         UBArray(iCol, iSize + 1) = CStr(RowBuf.Value(0, iCol))
      End If
   Next iCol
```

When a row is added to the Grid, the array must be re-dimensioned so that it has room to hold the extra data. The first part of this code stores the size of the array's second subscript (the number of "rows") and uses it to re-dimension the array to hold one more data record. Using the **ReDim Preserve** statement ensures that the data will be retained when the array is re-dimensioned.

Once the extra space has been added to the array, the data is transferred from the ssRowBuffer into the new row. Note the use of the `CStr` function to retrieve the data. Because the array is a String variable, you must be sure to pass it a String value.

10. The next step is to implement deletion of data from the array. There are three events that are used when deleting data; **BeforeDelete**, **UnboundDeleteRow** and **AfterDelete**. You must enter code in all three of these events to completely implement data deletion. The three events are necessary because the main Unbound event - **UnboundDeleteRow** - fires multiple times if multiple rows are deleted from the Grid. If you were to delete data from the array in this event, the indexes of the array would change and the bookmarks passed to the event would be inaccurate after the first deletion.

To avoid this problem, you can take the following approach. When the delete begins, you first calculate the size the array will be when the delete has finished. You then create a clone of the array. As each row is deleted, you flag the corresponding record in the clone array. Finally, after all the rows marked for deletion have been flagged in the clone array, you re-dimension the main array to its new size and copy the data from the clone array into the main array. As you copy the data, you skip any records in the clone array that have been flagged for deletion. Finally, you destroy the clone array.

Enter the following code in the **BeforeDelete** event:

```
iNewSize = UBound(UBArray, 2) - SSDBGrid1.SelBookmarks.Count
vClone = UBArray()
```

This code uses the two form-level variables that have been unused until now. vClone is a Variant type variable that holds the cloned array, and iNewSize represents the size the array will be when the data has been deleted. This code calculates the new size of the array by subtracting the number of selected rows from the current array size. Then it creates the clone array.

Add the following code to the **UnboundDeleteRow** event:

```
vClone(0, Bookmark) = "*SS.DELETED*"
```

This code tags each row in the cloned array with a text string that marks the row as one to be deleted.

Finally, add the following code to **AfterDelete** event procedure:

```
Dim iUBRow As Integer
Dim iCol As Integer
Dim iC As Integer

   ReDim UBArray(0 To (iCols - 1), 0 To iNewSize)
   iUBRow = 0

   'Copy data from the clone array into the
   'main array, skipping the deleted records
   For iC = 0 To UBound(vClone, 2)
      If vClone(0, iC) <> "*SS.DELETED*" Then
         For iCol = 0 To (iCols - 1)
            UBArray(iCol, iUBRow) = vClone(iCol, iC)
         Next iCol
         iUBRow = iUBRow + 1
      End If
   Next iC

   vClone = ""
   iCount = UBound(UBArray, 2)
   SSDBGrid1.Rows = iCount
   SSDBGrid1.Refresh
```

This code completes the deletion of the data. First, the main array is re-dimensioned to its new size (as established in the **BeforeDelete** event.) Then each row in the clone array is checked for the presence of the deleted flag. If the flag is not present, the data is copied into the main array. This is done until all the data in the clone array has been either copied or skipped. Finally, the clone array is destroyed, the global counter variable is updated to reflect the number of elements in the new array, and the Grid is updated with the information about the new number of rows and told to repopulate itself.

11. Finally, you must add code to handle the editing of the Grid data and the updating of the corresponding array elements. Add the following code to the **UnboundWriteData** event:

```
Dim iCol as Integer

    For iCol = 0 To (iCols - 1)
        If Not IsNull(RowBuf.Value(0, iCol)) Then
            UBArray(iCol, CInt(WriteLocation)) = _
            CStr(RowBuf.Value(0, iCol))
        End If
    Next iCol
```

This code iterates through each column in the array and replaces it with the value from the Grid from the current row. The current row is passed to the event as the *WriteLocation* parameter. When this event occurs, the ssRowBuffer contains only one row - the one with the updated data from the Grid.

You have now completed the second Unbound tutorial. You have expanded on the techniques you used in the first tutorial to link the DataGrid to a completely different kind of record source. You should now have a better understanding of how the Unbound events work and a more generalized sense of how to link the Grid to disparate data sources.

In the next exercise, you will see how to use the Grid in AddItem mode. In this mode, the Grid functions much like an extended List Box control. All data is stored internally in memory as soon as it is added to the AddItem Grid.

DBGrid: Exercise 4 (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.

DBGrid: Exercise 5 (Bookmarks)

This exercise explores the concept of bookmarks as they are used in the DataGrid, DataCombo and Data Dropdown controls. It will describe what bookmarks are, how to use them, how not to use them, and will hopefully give you some insight into how bookmarks can be made to work for you.

Please note that bookmarks are not specific to Data Widgets. The concept of bookmarks comes from the design of Microsoft's data access architecture. Recordsets and resultsets of DAO, RDO, and ADO all use bookmarks. While a complete and detailed discussion of bookmarks is beyond the scope of this topic, you will see how bookmarks are used with the Data Widgets controls. Most of this topic refers to the DataGrid control, but the same concepts usually apply to the DataCombo or DataDropDown as well.

Unlike the other tutorials, this one uses a question and answer format to cover as much material as quickly as possible. Code samples are included where appropriate.

What is a Bookmark?

1. *What a bookmark is:*

   The exact structure of a bookmark is unimportant and often confusing. What you must know about bookmarks is that a bookmark is a unique identifier for a row in a DataGrid, recordset, or resultset. Think of the bookmark as the address of a row or record. The actual value of the bookmark is not important.

2. *What a bookmark is not:*

   A bookmark is not a row number. Bookmarks are not displayable characters. They are not numbers or letters that you can show on the screen. If you try to display a bookmark, you will occasionally see numbers and letters, but these are not an accurate indication of the bookmark's contents nor are they reliable. It is never necessary to know what the value of a bookmark is or to display it.

3. *What data type is a bookmark?*

   In Visual Basic, if you wish to store a bookmark, you must store it in a variable that is a Variant Data Type. In Visual C++, you can use a COleVariant. On a web page, all variables are Variants, so any variable can hold a bookmark.

Why do I need bookmarks?

1. *Can't I just use row numbers?*

   For the most part, Data Widgets does not make use of absolute row numbers. In some circumstances, row numbers are available when referencing data, but in many cases they are not. It is almost always easier and more efficient to use bookmarks.

2. *What can I do with bookmarks?*

   Bookmarks can be used to do all the things that you would expect to do with row numbers. At first the use of bookmarks may not seem intuitive, but once you understand how they work, you will find they are more powerful and more flexible than row numbers.

   Here is a short list of some of the things you can do using bookmarks:

   - Move to a row in a DataGrid or a record in a recordset or resultset, or position to an item on the list of a DataCombo or DataDropDown.
   - Store the position of a particular record, so you can come back to it later.
   - Read values in a row of the DataGrid that is not the current row.
   - Read data from the selected rows in a DataGrid.
   - Determine if the user has changed rows or columns in the **RowColChange** event of the DataGrid.
   - Ensure that a particular row of the DataGrid is visible in the DataGrid.
   - Position a row in the DataGrid to the corresponding record in a recordset, or vice versa.

How do I use bookmarks?

1. *How do I get the bookmark of a row?*

   Knowing how to retrieve a bookmark is the most basic and important step in learning how to use them. A bookmark can be stored in a variable that is of type Variant, but how do you actually retrieve a bookmark? There are a number of ways.

   - ***The Bookmark Property***. The simplest and most basic way to get a bookmark is by using the **Bookmark** property of the DataGrid. This line of code takes the bookmark of the current row of the DataGrid and stores it in a variable called `vBkMark`, which is a Variant.
     ```
     vBkMark = SSDBDataGrid1.Bookmark
     ```

   - ***The GetBookmark Method***. The **Bookmark** property is fine if you want the bookmark of the current row, but what about bookmarks of the other rows in the DataGrid? Another way to get the bookmark of a row is to use the **GetBookmark** method. **GetBookmark** returns a bookmark when you call it with an integer value. The value you pass to **GetBookmark** is a relative offset from the current row of the DataGrid. For example, if you want the bookmark of the row immediately above the current row, you could use the following:
     ```
     vBkMark = SSDBDataGrid1.GetBookmark(-1)
     ```

     If you want to get the bookmark of the row immediately below the current row, you could use:
     ```
     vBkMark = SSDBDataGrid1.GetBookmark(1)
     ```

     If you want to get the Bookmark of the row 25 rows below the current row, you could use:
     ```
     vBkMark = SSDBDataGrid1.GetBookmark(25)
     ```

     Using this method, it is easy to get the bookmark of a row using an absolute row number by first moving to the first row in the DataGrid:
     ```
     SSDBDataGrid1.MoveFirst
     ```

     The DataGrid will now be on the very first row. Since **GetBookmark** gives you the bookmark of a row relative to the current row, any number you supply will be equivalent to the absolute row number.

So if you do this:

```
vBkMark = SSDBDataGrid1.GetBookmark(0)
```

You will now get the bookmark of the first row in the DataGrid (which would be row 0 if the DataGrid used absolute row numbers.) Calling **GetBookmark** with the number 20 would give you the bookmark of the twentieth row.

The disadvantage of this approach is that you must move the DataGrid to a new position. If the grid is currently in Edit Mode, this will cause the grid to attempt to commit the changes in the current row before moving to the first row.

- ***The RowBookmark Method***. You can also use the **RowBookmark** method to retrieve the bookmark of a row. **RowBookmark** accepts a visible row number as a parameter, and it returns the bookmark of that row. Note that this is *not* an absolute row number, it is only the number indicating where the row is currently displayed in the grid. This means that calling **RowBookmark** with the number 2 will always return the bookmark of the row second from the top of the grid, whether that row is the second record in the recordset, the fiftieth or the two-hundred and fiftieth. Every time you scroll the grid, the bookmark returned by **RowBookmark** for a given value will change.
  To retrieve the bookmark of the first visible row in a Grid, you would use the following code:

```
vBkMark = SSDBDataGrid1.RowBookmark 1
```

- ***The FirstRow Property***. Another property of the DataGrid that returns a bookmark is the **FirstRow** property. **FirstRow** returns or sets the bookmark of the first visible row. So this code will return exactly the same bookmark as the code in the example above.

```
vBkMark = SSDBDataGrid1.FirstRow
```

- ***The SelBookmarks Collection***. The DataGrid also uses bookmarks to keep track of which rows in the DataGrid are selected. Note that a selected row is not necessarily the same thing as the current row. The DataGrid may have many selected rows, but only one row can be the current row.
  When you select a row with the mouse (using the RecordSelectors, for example) or keyboard, the DataGrid adds the bookmark of that row to the SelBookmarks collection. There is probably no reason to do so since it is already stored for you, but as an example, suppose you wanted to store the bookmark of the first selected row. You could use this code to do it:

```
vBkMark = SSDBDataGrid1.SelBookmarks(0)
```

- ***The LastRow Parameter***. The **RowColChange** event of the DataGrid provides a *LastRow* parameter, which returns the bookmark of the previous row. A common use for this parameter is to determine if the event fired because the user changed rows or changed columns. You can determine this by comparing the current bookmark of the DataGrid with the *LastRow* parameter. If they are the same, the user has only changed columns and is still on the same row. If they are different, the user has changed rows.
  Because bookmarks are variants and cannot be compared directly, you must make allowances when dealing with this situation. Take the following code for example:

```
'The following line generates an error
If SSDBDataGrid1.Bookmark = LastRow Then
   Debug.Print "Same Row"
Else
   Debug.Print "Different Row"
End If
```

The code above will generate an error on the first line because comparing two bookmarks in this way is not possible. To get this to work correctly, both bookmarks must be converted to strings. Like so:

```
If CStr(SSDBDataGrid1.Bookmark) = CStr(LastRow) then
   Debug.Print "Same Row"
Else
   Debug.Print "Different Row"
```

```
End If
```

I've got my bookmark, now what do I do with it?

1. *Use the Bookmark Property*. Once you have a bookmark, you have the address of a particular row, so you can always access that row. You can use the bookmark to move the DataGrid to that row, like so:

   ```
   SSDBDataGrid1.Bookmark = vBkMark
   ```

   This line of code does not change the bookmark of the current row. It moves the DataGrid to the row whose bookmark is vBkMark.

2. *Use the FirstRow Property*. Normally, making a row the current row will scroll the DataGrid automatically to bring it into view. But suppose you want to bring the row into view without making it the current row. You could do this:

   ```
   SSDBDataGrid1.FirstRow = vBkMark
   ```

   It is not uncommon to store both the **Bookmark** and the **FirstRow** properties of the DataGrid, so that the DataGrid can later be restored to the same current row and same position as it was when you saved the information.

3. *Use the CellText Method*. It is easy enough to read values from the current row of the DataGrid, using the **Text** or **Value** property of the Column object. But what if you want to read values from a row other than the current one, without moving off the current row? You could do this using the **CellText** or **CellValue** methods of a Column object. These methods require you to pass them the bookmark of the row you want to read the data from. For example:

   ```
   Debug.Print SSDBDataGrid1.Columns(0).CellText(vBkMark)
   ```

   Or

   ```
   Debug.Print SSDBDataGrid1.Columns(0).CellValue(vBkMark)
   ```

   These two lines of code display the contents of the first field (Column 0) in the row whose Bookmark is contained in vBkMark.

4. *Use the SelBookmarks Collection*. Now that we know how to read data from a row by using its bookmark, it becomes easy to read data from all the selected rows in the DataGrid, using the SelBookmarks collection. The following code illustrates how to do this:

   ```
   Dim iC as Integer
   Dim vBkMark as Variant

      For iC = 0 to (SSDBDataGrid1.SelBookmarks.Count - 1)
         vBkMark = SSDBDataGrid1.SelBookmarks(iC)
         Debug.Print SSDBDataGrid1.Columns(0).CellValue(vBkMark)
      Next iC
   ```

   Conversely, you can use a bookmark to select a row. Simply add the bookmark of the row to the SelBookmarks collection like so:

   ```
   SSDBDataGrid1.SelBookmarks.Add vBkMark
   ```

5. *Use the Bookmark Property*. Bookmarks can be used to synchronize a Data Widgets control with a recordset or resultset object. For example, if you want to position a recordset to the same row as the DataGrid, you can use:

   ```
   Data1.Recordset.Bookmark = SSDBDataGrid1.Bookmark
   ```

   This assumes that the recordset bookmarks match up with the DataGrid's bookmarks. This will always be the case if the DataGrid is bound or unbound, but will not work in AddItem mode.

   You can do the reverse operation to position the DataGrid to the same row as the recordset.

   ```
   SSDBDataGrid1.Bookmark = Data1.Recordset.Bookmark
   ```

If the DataGrid is bound, this is unnecessary since moving the recordset will automatically position the DataGrid, but it is very useful for the DataCombo or DataDropDown control or for a DataGrid in unbound mode.

A Special Case - AddItem data mode

1. *The AddItemRowIndex Method*. There is a special case that involves using the DataGrid in AddItem mode. The AddItem data mode provides two additional methods to help you work with bookmarks. The first one is **AddItemRowIndex**. This method takes the bookmark of a row and returns the absolute row number:

```
Dim vBkMark as Variant
Dim lRowNum as Long

   vBkMark = SSDBDataGrid1.Bookmark
   lRowNum = SSDBDataGrid1.AddItemRowIndex(vBkMark)
```

2. *The AddItemBookmark Method*. The other method performs the opposite function and is called **AddItemBookmark**. You pass it an absolute row number and it returns the bookmark of the row.

```
vBkMark = SSDBDataGrid1.AddItemBookmark(50)
```

You should now have a basic understanding of bookmarks and how they work. Of course, most real applications will require more complex code than is provided here, but the samples above should give you a good understanding of what methods and properties you can use to accomplish various bookmark-related tasks.

DBGrid: Exercise 6 (StyleSets and the RowLoaded Event)

Although the Data Widgets controls offer many formatting options for controlling the appearance of your application, one of the most powerful formatting mechanisms is the use of StyleSets. StyleSets give you a way to define a consistent set of formatting options and apply them to different parts of the control at different times, as circumstances merit.

StyleSets are available in the DataGrid, DataCombo and Data Dropdown controls. They can be used to format the parts of the control which label data, such as column and group headers. They can also be used to format data within the control in the form of columns, rows and cells. The method you use to apply StyleSets varies depending on which part of the control you are applying them to. Applying a StyleSet to a static part of the control such as a column or group header is as simple as setting a property value. Applying StyleSets to the more dynamic parts of the control, such as rows and cells, is a bit more involved and requires utilizing the **RowLoaded** event of the control.

This tutorial covers two basic concepts. One is the use of StyleSets to apply formatting to the various parts of the DataGrid. (The techniques covered are also applicable to the DataCombo and DataDropdown.) The other is the use of the **RowLoaded** event to apply formatting to rows and columns, as well as extending the flexibility of the Grid in general.

Background

A StyleSet is an object in a Data Widgets control (DataGrid, DataCombo or DataDropdown) that stores a set of formatting attributes. You can create a StyleSet at design time using the Grid Editor or at run time by simply giving it a name and assigning values to its properties. To use a StyleSet, you assign it to an object in the grid - a column, column header, group, group header, row, cell or other visible part of the control.

Each StyleSet object is a member of the StyleSets collection. When you add a StyleSet object to the collection, you give it a descriptive name, such as "HighContrast" or "FallColors." Whenever you need to use the StyleSet, you refer to it by it's descriptive name. (Although StyleSet objects in the collection do have numeric indexes, use of them to refer to individual objects is not recommended.)

StyleSets are applied to an object by setting a property of the object to the descriptive name of an existing StyleSet. Some objects have more than one property that can be set to a StyleSet. The property you use determines which part of the object the StyleSet's formatting will be applied to. For example, a Column object has a **StyleSet** property and a **HeadStyleSet** property. Setting the first one applies formatting to the column's data; the second formats the column's heading.

The StyleSet object has properties that mirror those of the other objects in the control that can be formatted. For

example, a StyleSet object has a **BackColor** property, a **ForeColor** property, a **Font** property, a **Picture** property, and so on. Because a StyleSet object does not correspond to a visible part of the control by itself, setting these properties may not have any immediate effect on the way the control looks. However, if the StyleSet has already been applied to a visible object or objects, changing these values will change the appearance of one or more visible parts of the control.

Once you have created a StyleSet object and assigned values to its properties, you can then assign those property values to different parts of the control by applying the StyleSet to the appropriate object. If you later change a property in the StyleSet, any part of the grid that uses that StyleSet will reflect the change. For example, if you assign `vbYellow` to the **BackColor** property of a StyleSet, then apply the StyleSet to a column header, the background of the column header will become yellow. If you then change the property of the StyleSet to `vbGreen`, the background of the column header will change to green the next time the column header is repainted.

Changes to StyleSets are not applied until a repaint occurs in the part of the Grid affected by the change. You may want to refresh the Grid display whenever you change a StyleSet setting to make sure your changes are displayed immediately.

Exercise

1. Place an SSDBGrid control directly on the form by double clicking on the  tool in the Visual Basic toolbox. Resize the Grid so that it occupies the top half of the form..
2. Set the **DataMode** property of the Grid to '2 - AddItem'.
3. In order to observe the effects of StyleSets, you must have objects such as columns, groups, rows and cells to apply them to. This first section of code simply sets up the Grid with columns and groups, then fills it with dummy data. Enter the following code in the **InitColumnProps** event of the Grid:

```
Dim iC As Integer
Dim iCol As Integer
Dim iCols As Integer
Dim sRowString As String

   iCols = 4
   sRowString = ""

   'Add columns to the grid
   For iCol = 0 To (iCols - 1)
      SSDBGrid1.Columns.Add iCol
   Next iCol

   SSDBGrid1.FieldDelimiter = "!"
   SSDBGrid1.FieldSeparator = ";"

   'Fill the  grid with dummy data
   For iC = 0 To 29
      For iCol = 0 To (iCols - 1)
         sRowString = sRowString & "!" & "Column " & iCol _
         & ", Row " & iC & "!;"
      Next iCol
      SSDBGrid1.AddItem sRowString
      sRowString = ""
   Next iC

   For iC = 0 To 1
      'Add groups to Grid
      SSDBGrid1.Groups.Add iC
      SSDBGrid1.Groups(iC).Caption = "Group " & iC
      SSDBGrid1.Groups(iC).Width = Int(SSDBGrid1.Width / 2)

      'Put columns in groups
```

```
      Select Case iC
      Case 0
         SSDBGrid1.Columns("Column 1").Group = iC
         SSDBGrid1.Columns("Column 1").Width = _
         Int(SSDBGrid1.Groups(iC).Width / 2)

         SSDBGrid1.Columns("Column 0").Group = iC
         SSDBGrid1.Columns("Column 0").Width = _
         Int(SSDBGrid1.Groups(iC).Width / 2)
      Case 1
         SSDBGrid1.Columns("Column 3").Group = iC
         SSDBGrid1.Columns("Column 3").Width = _
         Int(SSDBGrid1.Groups(iC).Width / 2)

         SSDBGrid1.Columns("Column 2").Group = iC
         SSDBGrid1.Columns("Column 2").Width = _
         Int(SSDBGrid1.Groups(iC).Width / 2)

   End Select
   Next iC
```

The **InitColumnProps** event should always be used for any code that creates or changes the layout in the grid before it is displayed. While you could put this code in the **Load** event of the form, it is safer and more efficient to use the **InitColumnProps** event. This code sets up the Grid with 30 rows and four columns of dummy data and creates a simple layout consisting two groups with two columns in each one.

4. Add a ListBox and a command button to the form in the area below the Grid. Label the command button "Create StyleSets" and add the following code to its **Click** event:

```
SSDBGrid1.StyleSets.RemoveAll
List1.Clear

SSDBGrid1.StyleSets.Add "Blues"
SSDBGrid1.StyleSets.Add "Inverse"
SSDBGrid1.StyleSets.Add "Warning"
SSDBGrid1.StyleSets.Add "XRay"

With SSDBGrid1.StyleSets("Blues")
   .Font.Name = "Arial"
   .Font.Bold = True
   .Font.Size = 10
   .BackColor = vbBlue
   .ForeColor = vbCyan
   .Picture = LoadPicture("blues.bmp")
End With
List1.AddItem "Blues"

With SSDBGrid1.StyleSets("Inverse")
   .Font.Name = "MS Sans Serif"
   .Font.Bold = False
   .Font.Italic = False
   .Font.Size = 8
   .AlignmentText = ssCaptionAlignmentRight
   .BackColor = vbBlack
   .ForeColor = vbWhite
End With
List1.AddItem "Inverse"

With SSDBGrid1.StyleSets("Warning")
   .Font.Name = "Arial"
   .Font.Bold = True
```

```
    .Font.Italic = True
    .Font.Size = 12
    .BackColor = vbRed
    .ForeColor = vbYellow
    .Picture = LoadPicture("warning.bmp")
End With
List1.AddItem "Warning"

With SSDBGrid1.StyleSets("XRay")
    .Font.Name = "Times New Roman"
    .Font.Italic = True
    .Font.Size = 8
    .AlignmentText = ssCaptionAlignmentRight
    .AlignmentPicture = ssPicAlignRightofText
    .BackColor = vbBlack
    .ForeColor = vbGreen
    .Picture = LoadPicture("xray.bmp")
End With
List1.AddItem "XRay"

List1.ListIndex = 0
Command1.Enabled = False
```

This code first clears the StyleSets collection, removing any StyleSet objects that were previously defined in the Grid. It then creates four new StyleSets - Blues, Inverse, Warning and XRay - by simply adding their names to the collection. Once the StyleSets have been created, each one is assigned a set of attributes and its name is added to the list box.

Some of the StyleSets contain only a few formatting options, such as text size and color. Others contain more advanced settings, such as pictures and alignment information. As you can see, StyleSets are always referred to by their name rather than their index value.

You may need to change the **LoadPicture** statements to point to the actual location of the sample picture files. The pictures are included with the sample project for this exercise and can be found in the `SAMPLES/CHAP05/ROWLOADED` subdirectory of the directory where Data Widgets is installed.

At the end of the code, the command button is disabled. Although it might seem that this button could be used multiple times (since the beginning of the **Click** event removes all StyleSets and clears the list) clicking the button will produce an error if StyleSets have been assigned to any part of the Grid. Once a StyleSet is in use, you cannot remove it from the collection.

5. Once the StyleSets are created, you can apply them to different parts of the grid and observe their effects. Create a second Command button on the form and label it "Apply StyleSets". Add the following code to the **Click** event of the new command button:

```
If List1.ListCount = 0 Then Exit Sub

If SSDBGrid1.HeadStyleSet <> "" Then
   SSDBGrid1.HeadStyleSet = ""
   SSDBGrid1.Groups(1).HeadStyleSet = ""
   SSDBGrid1.Groups(1).StyleSet = ""
   SSDBGrid1.Columns(0).HeadStyleSet = ""
   SSDBGrid1.Columns(0).StyleSet = ""
End If
If SSDBGrid1.Groups(1).HeadStyleSet <> "" Then
   SSDBGrid1.HeadStyleSet = "Warning"
End If
If SSDBGrid1.Groups(1).StyleSet <> "" Then
   SSDBGrid1.Groups(1).HeadStyleSet = "Blues"
End If
If SSDBGrid1.Columns(0).HeadStyleSet <> "" Then
   SSDBGrid1.Groups(1).StyleSet = "XRay"
```

```
End If
If SSDBGrid1.Columns(0).StyleSet <> "" Then
   SSDBGrid1.Columns(0).HeadStyleSet = "XRay"
End If
SSDBGrid1.Columns(0).StyleSet = "Inverse"
```

Run the project, click the first button to create they StyleSets, then click the second button a number of times. You will see StyleSets applied in succession to different parts of the Grid each.

This code applies different StyleSets to different parts of the Grid depending on how many times the button has been clicked. First, a test is performed to see if the StyleSets have been created, and if they haven't no action is taken.

The first time through the procedure, no StyleSets have been assigned yet, so all the `If` tests fail and the "Inverse" StyleSet is applied only to the first column in the Grid. This is done by assigning the StyleSet's name to the column's **StyleSet** property. Assigning a StyleSet to a column formats the data in that column. Notice that the StyleSet overrides the odd and even row colors specified by the Grid.

The second time through the code, the final `If` statement is True, and the 'XRay" StyleSet is assigned to the **HeadStyleSet** property of the same column. This property controls the appearance of the column header. Notice that the picture appears in the header to the right of the text, and that the text is not aligned to the left.

The third time through the code, the second-to-last `If` statement is also true, and the "XRay" StyleSet is assigned to the **StyleSet** property of the second group. This affects the formatting of all the column data in that group.

Notice that, while the colors and fonts have been applied from the StyleSet, the alignment and picture settings have not. You cannot put pictures in cells using a column StyleSet, however you will see later how you can assign pictures to cells using the **RowLoaded** event.

The fourth time through the code, the "Blues" StyleSet is assigned to the **HeadStyleSet** property of the same group. Note that the setting for the group also affects the headings of the columns in that group, which do not have their own StyleSet settings. In this case, the picture from the StyleSet is applied to both the group and column headings. This is because columns inherit the properties of the group that contains them.

The fifth click of the button assigns the "Warning" StyleSet to the **HeadStyleSet** property of the Grid itself. Notice that this affects not only the Grid heading, but the record selectors and any group or column headings that do not have a StyleSet assigned to them. Groups and columns that already have their own StyleSet are not affected by the change. Notice also that the picture from the StyleSet is applied only to the Grid heading; group and column headings do not display the picture.

Clicking the button a sixth times removes all the StyleSet by assigning a null string to the relevant properties. Once the StyleSets have all been cleared, the "Inverse" StyleSet will again be assigned to the first column's data.

6. You can also use StyleSets to apply formatting to rows or even individual cells. Through code, you can even apply formatting based on the contents of a particular cell. You use StyleSets to apply this formatting, usually in conjunction with the **RowLoaded** event of the grid.

   However, the **RowLoaded** event is not always necessary if you want to format Grid data. You have seen that a StyleSet applied to the Column or Group object will format the data in that column or group using the StyleSet's attributes. There are two other means by which you can format the appearance of cells without using **RowLoaded**. One is to set the **ActiveRowStyleSet** property of the grid, and the other is to set the **StyleSet** property of the ActiveCell object

   Add the following code to the Click event of the first command button (the one that creates the StyleSets):

   ```
   SSDBGrid1.ActiveRowStyleSet = "Blues"
   SSDBGrid1.ActiveCell.StyleSet = "XRay"
   ```

   Run the Project and click the "Create StyleSets" button. Use your mouse to click on different cells in the Grid. You will see that the current row is always formatted with the attributes of the "Blues" StyleSet, while the active cell uses the font, color, alignment and picture information of the "XRay" StyleSet.

7. The **RowLoaded** event fires whenever a row of the grid must be displayed. You can take advantage of this to assign formatting to each row "on the fly" when the Grid calls the event. Add the following code to the **RowLoaded** event of the Grid:

```
Dim sCellText As String

   If List1.ListCount = 0 Then Exit Sub

   sCellText = SSDBGrid1.Columns(1).Text
   If Right(sCellText, 1) = "0" Or _
   Right(sCellText, 1) = "5" Then
      SSDBGrid1.Columns(1).CellStyleSet ("Inverse")
   End If
```

Run the project, click the "Create StyleSets" button, then scroll down through the Grid to see the effect of the code in this event. This code applies the Inverse StyleSet to any cell in the second column that ends in a zero or a five. In this code, the **Text** property of the Column object returns the text of the row being currently loaded. (This only occurs in the **RowLoaded** event.) This text is examined to see if it meets the criteria, and if it does, the **CellStyleSet** method of the column is used to assign a StyleSet to the cell.

Certain properties, such as **Text** and **Value**, function slightly differently in this event. The **CellStyleSet** method also functions differently here than it does outside of this event. For example, the **CellStyleSet** method usually requires you to pass it a visible row number to identify the row containing the cell. Here, none is required. Also, the **Text** property usually returns information about cells in the current row. Here, it applies to the cell in the row being loaded, regardless of whether or not it is the current one.

This concludes the StyleSet and RowLoaded tutorial. You have seen how powerful and convenient StyleSets can be, and you have learned how to apply them to different parts of the Grid. You have also seen how the RowLoaded event can be used to format rows and cells "on the fly" as they appear in the Grid.

In the next exercise, you will learn all about the DataGrid's ability to export data into HTML for use on the Internet or a corporate intranet.

DBGrid: Exercise 7, Part I - Exporting HTML

In this exercise, you will see how to export data from the Data Grid control into an HTML file that can be placed directly on a corporate intranet or the World Wide Web and viewed with any web browser. You will start with the simplest forms of exporting and move on to advanced multi-file exports used to create master/detail relationships. Along the way, you will also see how to use the layout saving feature of the grid.

There are four parts to this tutorial:

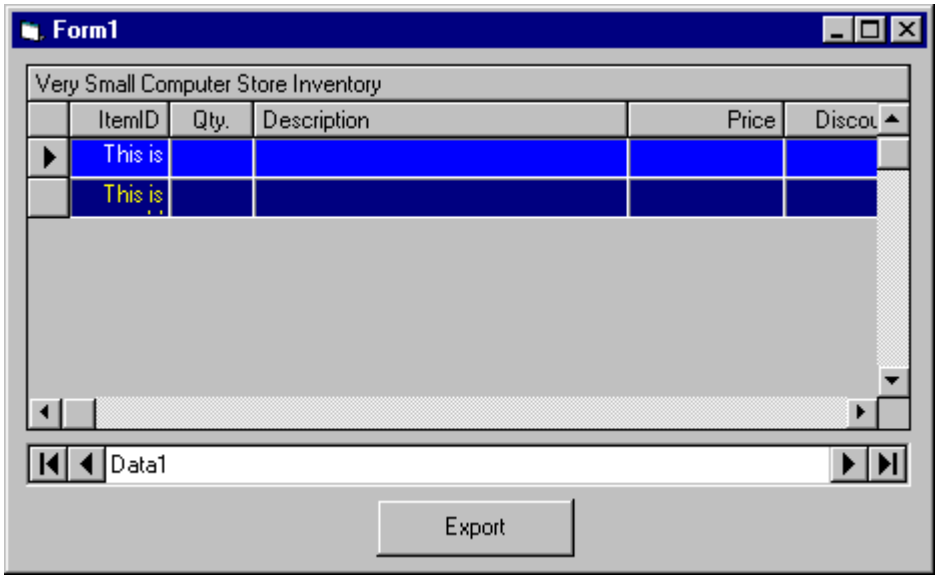
- **Basic HTML Exporting**
- **Exporting Grid Formatting**
- **Exporting Row Data**
- **Creating a Master/Detail Export**

Note To complete this exercise, you will need a number of files that are located in the SAMPLES/CHAP05/HTML directory, located under the directory where you installed Data Widgets. The project files created in this tutorial are also located in this directory in individual folders, according to which part of the exercise they are used with.

1. Create a new Visual Basic project, and add the Data Grid control to the project.
2. On a new form, add a Data Grid, a Visual Basic data control, and a command button.
3. Set the **DatabaseName** property of the data control to `SMALL.MDB`, which is located in the SAMPLES/CHAP05/HTML subdirectory of the directory where you installed Data Widgets. Set the **RecordSource** property of the control to "Stock."
4. Set the **DataSource** property of the grid to Data1. This will bind the grid to the database, which contains a few records that represent the inventory of a very small computer store.
5. In the past, the next step would be to set up the grid columns using the Grid Editor. But because Data Widgets 3.1 includes the ability to save and restore layouts, you can simply load a file that was created for you with the formatting already done.

   Right-click on the grid to bring up the context menu, then select Properties. The property pages for the grid will appear. Click the "Load..." button, and when the file name dialog appears, click the "..." button to invoke a File Open common dialog. Locate the file `STOCK.GRD` (located in the same directory as `SMALL.MDB`) and

open it. Then press "OK" in the Restore File Name dialog to complete your selection. Finally, click "OK" to accept your changes and close the property pages. Your form should look something like this:



The grid is now completely laid out and formatted, and you are ready to proceed to the next step.

6. You are going to add some code to detect when an item is out of stock and change its appearance in response. Add the following code to the **RowLoaded** event of the Grid:

```
Dim iC as Integer

If SSDBGrid1.Columns(1).CellValue(Bookmark) = 0 Then
   For iC = 0 To (SSDBGrid1.Columns.Count - 1)
      SSDBGrid1.Columns(iC).CellStyleSet "Reorder"
   Next iC
End If
```

This code checks the value in the Quantity column and, if it is zero, applies the "Reorder" StyleSet to every cell in the row. The "Reorder" StyleSet was stored in the `STOCK.GRD` layout file, and was added to the grid when you loaded that file. It makes text appear in red, bold italics.

7. Change the **Caption** of the command button to "Export" and specify the following code for its **Click** event:

```
SSDBGrid1.Export ssExportTypeHTMLTable, ssExportAllRows _
OR ssExportOverwriteExisting, "INSTOCK.HTM", "STOCKTPL.HTM"
```

This line of code performs the actual export to HTML. The constants specified for the **Export** method indicate that you want to export data as a full HTML table (*ssExportTypeHTMLTable*) as opposed to a text file or individual rows of HTML data. You will be exporting all the rows in the grid (*ssExportAllRows*) and you want to overwrite any existing file with the name you've specified (*ssExportOverwriteExisting*). The control will generate an HTML file called `INSTOCK.HTM` and will use the file `STOCKTPL.HTM` as a template to create that file.

It is the HTML templates that provide the flexibility behind the HTML export capabilities of Data Widgets. By formatting the HTML template using a variety of options, you gain fine control over the appearance of the generated output.

Note You may need to specify the full path to the HTML files. If you receive a run-time error when attempting to invoke the **Export** method, specify fully qualified path names for the export and template file parameters and try again.

8. Next, you must create the `STOCKTPL.HTM` template file, which Data Widgets will use when generating the target HTML file (`INSTOCK.HTM`). Even if you have no experience with HTML you should not have any

problems following along, as the HTML used in this tutorial is quite simple.

Note If you wish to do advanced formatting using templates, a working knowledge of HTML is essential. If you want to learn more about creating HTML, many resources are available. Most bookstores have a variety of references on creating HTML, and there are free interactive guides on the World Wide Web that provide useful information. A good reference for beginners is the *NSCA Beginners Guide to HTML* at `http://www.ncsa.uiuc.edu/general/internet/www/htmlprimer.html`. A more comprehensive reference is *Yale University's C/AIM Web Style Guide* at `http://info.med.yale.edu/caim/manual`.

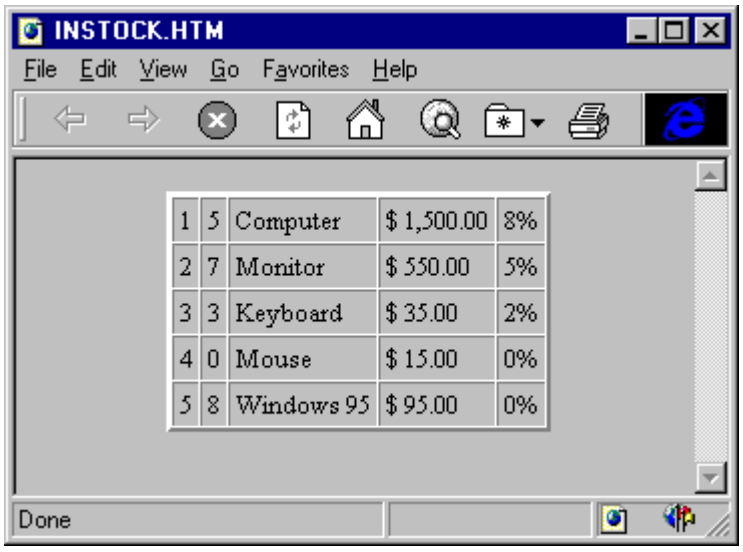
HTML files are simply text files that contain HTML formatting tags. For the purposes of this tutorial, you can use Notepad or any pure text editor to create the HTML templates. You may also use an HTML editor of your choice, provided it is not a WYSIWYG editor that hides the formatting tags from you or does not allow you to manually edit formatting information. If your HTML editor does not give you direct access to the HTML tags, use Notepad instead for this tutorial.

Open a new text file and enter the following lines of HTML:

```
<HTML>
<BODY>
<CENTER>{SSREPLACE TABLE DATA}</CENTER>
</BODY>
</HTML>
```

This is one of the simplest template files you can create. The real work is done by the {SSREPLACE} token. When Data Widgets creates the output file, the {SSREPLACE...} section of the template file will be replaced with data from the Data Grid control. In this example, you have indicated that the control should create a table structure based on the grid (TABLE) and fill it with the actual grid data (DATA). The generated HTML will be inserted between the <CENTER> tags, so the final result will be a web page with a centered table containing the grid data.

9. Save the text file as `STOCKTPL.HTM`. The file should reside in the location specified in your code for the **Export** method. Return to your Visual Basic project and run it. You should see the grid populated with data from the database.
10. Click the "Export" button on your form to generate a web page based on the data displayed. A file called `INSTOCK.HTM` will appear in the location you have specified. Open this file in your web browser. The output should look something like the following:



Data Widgets replaced the {SSREPLACE} token with grid data and created a very simple table. Nothing but the data is included. No formatting information from the grid is applied - only formatting applied within the HTML template (such as centering) is evident. If you are familiar with HTML, you may want to open the `INSTOCK.HTM` file in Notepad and look at the code that was generated. You will see something like the following:

```
<HTML>
```

```
<BODY>
<CENTER><TABLE BORDER=3 CELLPADDING=3 CELSPACING=0>
<TR>
<TD>1</TD>
<TD>5</TD>
<TD>Computer</TD>
<TD>$ 1,500.00</TD>
<TD>8%</TD>
</TR>
<TR>
<TD>2</TD>
<TD>7</TD>
<TD>Monitor</TD>
<TD>$ 550.00</TD>
<TD>5%</TD>
</TR>
.
.
.
</TABLE>
</CENTER>
</BODY>
</HTML>
```

(Some line breaks may show as black boxes in Notepad)

The {SSREPLACE} token has been replaced with a complete <TABLE></TABLE> structure. Each row in the grid has been delimited as a table row using <TR></TR> tags, and individual cells have been placed into rows as table data <TD></TD>.

DBGrid: Ex. 7, Part II - Exporting HTML

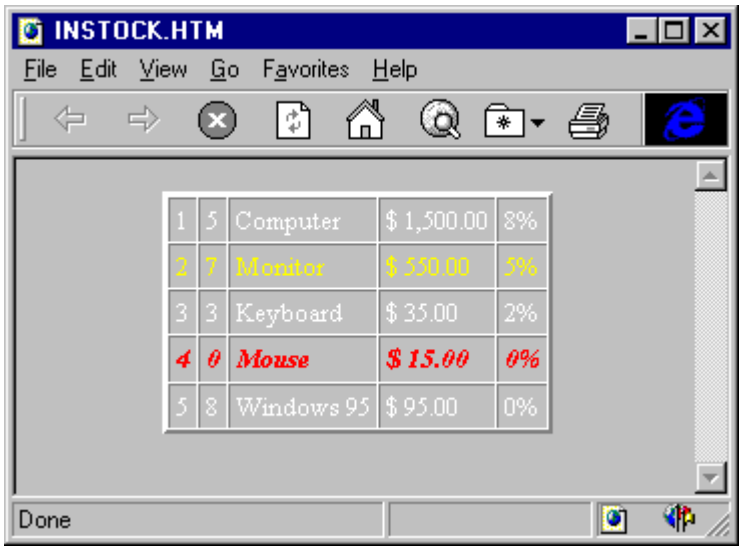
Now you know how to export basic grid data into HTML format. This is a useful capability if you simply need a raw dump of grid data, or want a simple HTML file that you can tweak by hand. But in most cases, you will want your HTML documents to be formatted in some way. You may have grids that you have set up with elaborate formatting - groups, column headers, custom colors and pictures, etc. You can easily generate HTML code that makes use of all these features; all you have to do is modify the template.

You have already seen how to use the TABLE and DATA attributes of the {SSREPLACE} token. In this part of the exercise, you will learn how to use many more attributes to customize the template. If you want to see a complete listing of the attributes you can use with the {SSREPLACE} token, consult the section on HTML Template Codes.

1. Open the `STOCKTPL.HTM` file in Notepad and add the following information to the {SSREPLACE} token:

   ```
   {SSREPLACE TABLE DATA B I COLOR}
   ```

2. Save the file and click on the "Export" button in your program. Open the resulting `INSTOCK.HTM` file in your web browser. The results should look something like this:



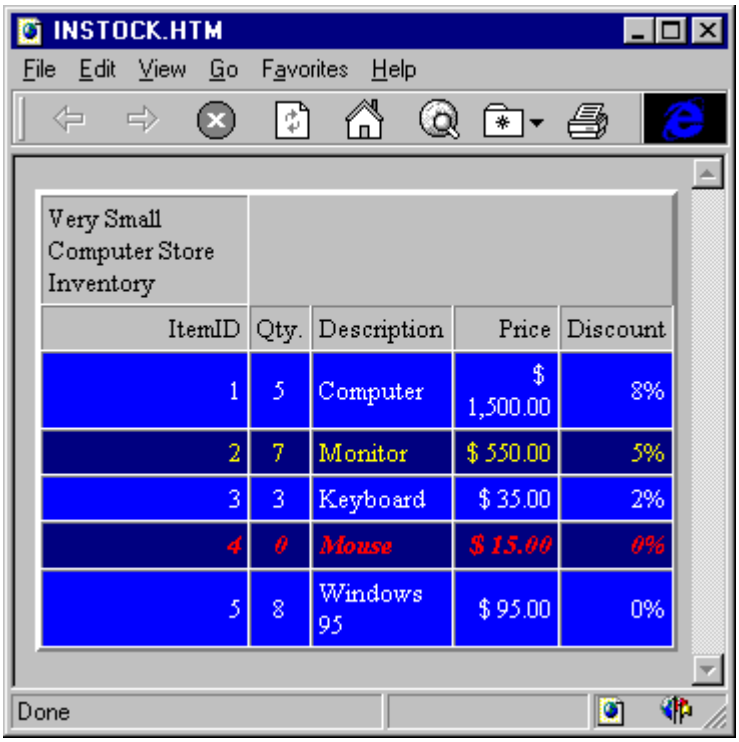
By adding the "B", "I" and "COLOR" attributes to the {SSREPLACE} token, you have instructed Data Widgets to include boldface (B), italic (I) and color (COLOR) information from the grid in the generated HTML. The {SSREPLACE} token supports dozens of attribute codes that you can use to control what will be included from the grid in the final printout.

3. Open the `STOCKTPL.HTM` file and append the following attributes to the {SSREPLACE} token:

   ```
   {SSREPLACE... CAPTION COLHEAD ALIGN BGCOLOR}
   ```

   The added attributes tell Data Widgets to include more data in the generated HTML table: the grid caption (CAPTION), the text of the column headers (COLHEAD), the alignment of the data in cells (ALIGN), and the background color of the cells (BGCOLOR). Note that some of these attributes are only applicable because you have instructed Data Widgets to generate a table (TABLE) for you, and might not be used if you were only pulling data from the grid and embedding it in your own formatted HTML. (You will see how to do this later in the exercise.)

4. Save the template file, then click the "Export" button in your application. Open the `INSTOCK.HTM` file in your web browser. You should see something like this:



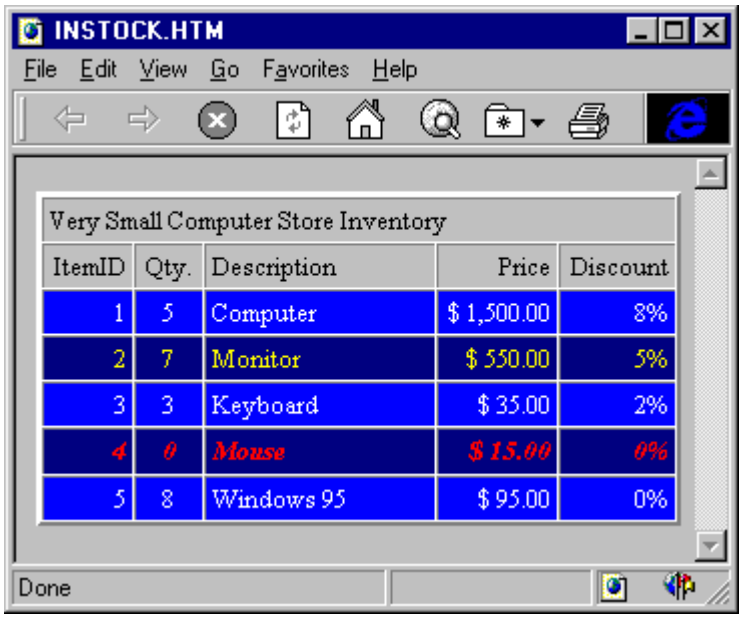
You can see that most of the attributes of the original grid have been exported to the HTML page. Some of the formatting is not quite right, due to the fact that the grid caption is confined to a single column, and some of the cells take up two lines. You can easily remedy these problems using another {SSREPLACE} attribute..

5. You can specify additional attributes that control the wrapping of text and the spanning of multiple columns in the table. For example, you probably want the grid caption to span all the columns of the table, as it does in the Data Grid. Modify the `STOCKTPL.HTM` file to append the following attribute to the {SSREPLACE} token:

```
{SSREPLACE...  WIDTH COLSPAN}
```

The WIDTH attribute will base the width of the table cells on the width of the corresponding grid columns. The COLSPAN attribute enables a single table cell (such as the cell containing the grid caption) to span multiple columns. The COLSPAN attribute can only be used if Data Widgets is generating the entire table (i.e. the TABLE attribute has been specified).

Save the HTML template file and click the "Export" button in your application. Then open the `INSTOCK.HTM` file in your web browser. The results should look something like this:



Now you have seen how to export various types of grid formatting into HTML format using the various attributes of the {SSREPLACE} token. Along the way, you have seen how to construct a single {SSREPLACE} token that exports the entire grid into HTML, with formatting information intact.

DBGrid: Ex. 7, Part III - Exporting HTML

At this point, you can easily export the contents of a Data Gird into HTML, with or without formatting. However, there may be situations where you do not want to export an entire grid's worth of data. Sometimes exporting only a single row or several rows may be sufficient.

In this part of the exercise, you will see how to export individual rows of data from the grid as distinct HTML files. This technique will become especially useful in the final section, when you will use it to create a master/detail series of web pages.

Note As with the previous exercises, in your code you may need to specify fully qualified path names in place of the "simple" file names shown here.

1. If your Visual Basic project is still running, stop it. Add the following line of code to the **Click** event of the "Export" button in your application, following the existing line of code:

```
SSDBGrid1.Export ssExportTypeHTMLRowFiles, ssExportCurrentRow _
OR ssexportoverwriteexisting, "ITEM.HTM", "1-ROWTPL.HTM", "ItemID"
```

(There should be two lines of code in the **Click** event when you are done. The first line, which you entered in Part I, will still generate the completely formatted table when the button is clicked.)

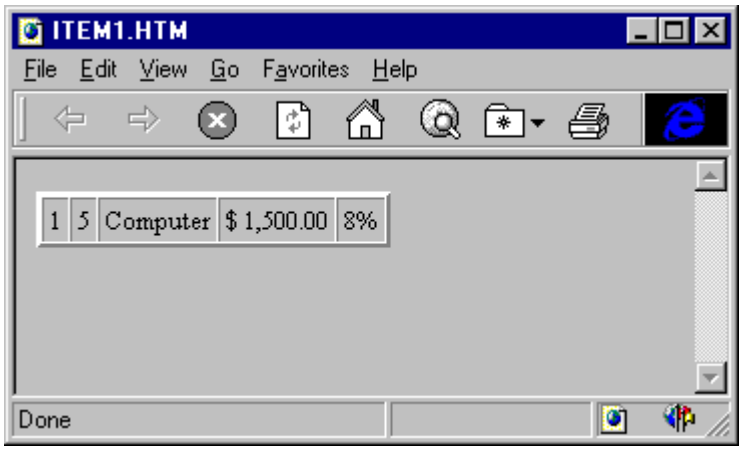
This new line of code will generate an HTML file based on one row - the row containing the currently active cell (*ssExportCurrentRow*). It will use the file `1-ROWTPL.HTM` as the template to generate the HTML. The name of the HTML file that is generated will be based on two pieces of information. The first is the file name specified for the *ExportToFile* parameter (`ITEM.HTM` in this case.) The control will take the first four letters of this filename as the beginning of the generated file's name. It will then concatenate the value of the field you specified in the *OutputFileField* parameter to complete the filename (in this case, the value of the "ItemID" field in the database.) Any existing file with the same name will be overwritten (*ssExportOverwriteExisting*).

2. Now you must create the `1-ROWTPL.HTM` template file. Open your HTML editor and specify the following:

```
<HTML>
<BODY>
{SSREPLACE TABLE DATA}
</BODY>
```

```
</HTML>
```

3. Save this file as `1-ROWTPL.HTM`, then run your application and click the "Export" button. Depending on which cell was currently selected when you clicked "Export", a new file will be created with a name like `ITEM1.HTM`. The file should look something like this:



4. Stop your Visual Basic project and change the line you added to read:

```
SSDBGrid1.Export ssExportTypeHTMLRowFiles, ssExportSelectedRows _
OR ssExportOverwriteExisting, "ITEM.HTM", "1-ROWTPL.HTM", "ItemID"
```

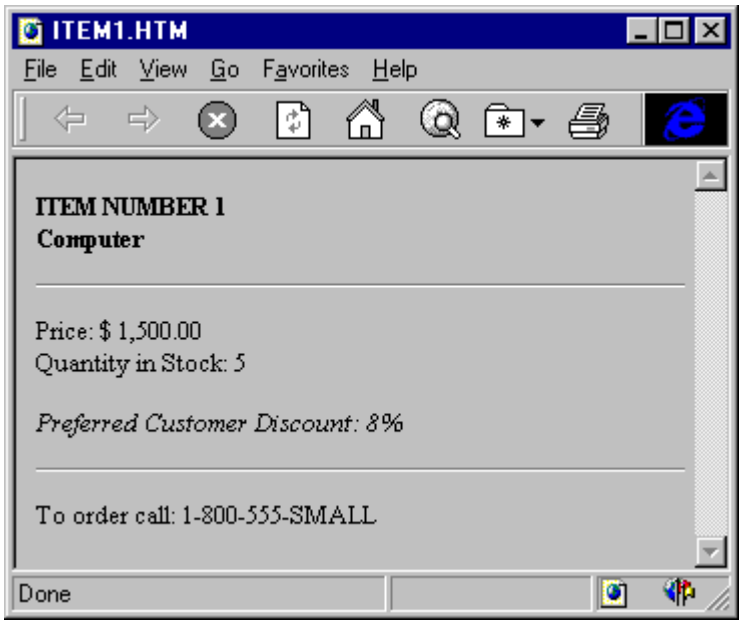
Then run the project again. Select the first three rows in the grid, then click the "Export" button. Because you specified that the **Export** method should export selected rows, and you have selected three rows in the grid, you will see three files created: `ITEM1.HTM`, `ITEM2.HTM` and `ITEM3.HTM`. Each file will contain a table similar to the one above, but with the data for an individual row.

5. Although it is easy to generate simple tables based on row data, there is much more you can do using the row export feature. Open the `1-ROWTPL.HTM` file in your HTML editor. Delete the existing HTML code and insert the following:

```
<HTML>
<BODY>
<B>ITEM NUMBER {SSREPLACE DATA FIELD="ItemID"}<BR>
{SSREPLACE DATA FIELD="Description"}</B><BR>
<HR>
Price: {SSREPLACE DATA FIELD="Price"}<BR>
Quantity in Stock: {SSREPLACE DATA FIELD="StockQty"}<BR>
<BR>
<I>Preferred Customer Discount: {SSREPLACE DATA
FIELD="Discount"}</I><BR>
<HR>
To order call: 1-800-555-SMALL
</BODY>
</HTML>
```

This HTML presents the information from the grid in a more textual manner. It pulls data from the grid and mixes it with text and formatting tags to generate a custom HTML page describing the item. By using multiple {SSREPLACE} tokens with the FIELD attribute, you can place data from the grid anywhere on the page.

Save `1-ROWTPL.HTM`, select the first three rows in the grid and click the "Export" button. Three new files will be created. Open the `ITEM1.HTM` file in your web browser. It will look something like this:



When you are ready, stop your Visual Basic project and close your web browser.

6. Now that you know how to export rows one at a time, you may find that you need to examine the data being exported and take some action based on its value. Earlier, you used the **RowLoaded** event to examine the value of a column and format the grid data based on the results. You can do the same thing to exported grid data using the **RowExport** event. This event gives you the chance to examine grid output and make changes while the data is being exported. It also gives you the opportunity to cancel the export once it is in progress. (Note that **RowLoaded** is also fired as each row is exported, so any formatting you do in that event is included in the export.)

   First, modify the second line of the "Export" button's **Click** event so that it contains the following code:

```
SSDBGrid1.Export ssExportTypeHTMLRowFiles, ssExportAllRows _
OR ssExportOverwriteExisting, "ITEM.HTM", "1-ROWTPL.HTM", "ItemID"
```

7. Open `1-ROWTPL.HTM` in your HTML editor and make the following change. Add an extra {SSREPLACE} tag as shown. Replace the following:

```
Quantity in Stock: {SSREPLACE DATA FIELD="StockQty"}<BR>
<BR>
<I>Preferred Customer Discount: {SSREPLACE DATA
FIELD="Discount"}</I><BR>
```

   With the following:

```
Quantity in Stock: {SSREPLACE DATA FIELD="StockQty"}<BR>
{SSREPLACE TAGVAR}<BR>
<I>Preferred Customer Discount: {SSREPLACE DATA
FIELD="Discount"}</I><BR>
```

   The TAGVAR attribute causes the token to be replaced with the **TagVariant** property of the Grid control in the generated file. In the next step, you will take advantage of this feature to expand your customization of the generated HTML.

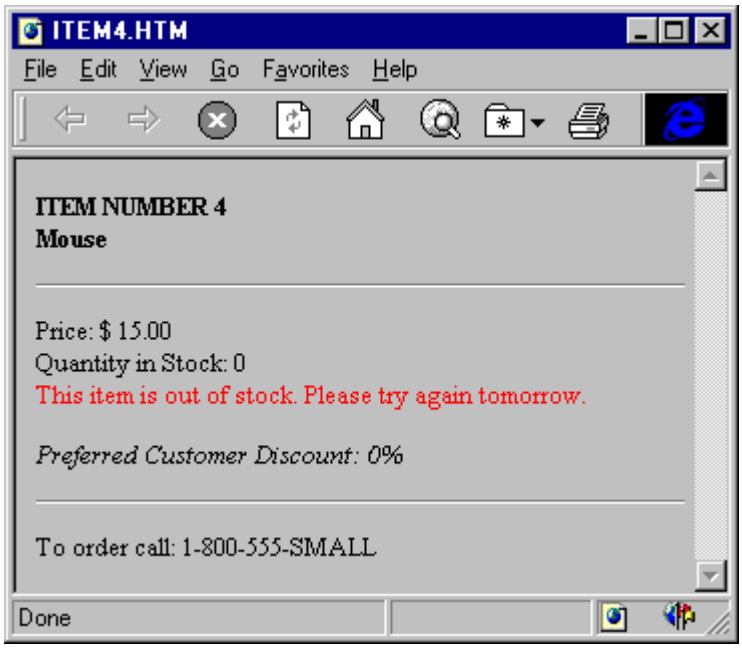
8. Select the **RowExport** event of the Data Grid and enter the following code. (Because the following HTML requires embedded quotation marks, the ASCII value of the quotation mark character - Chr$(34) - is used to construct the string.)

```
If SSDBGrid1.Columns(1).CellValue(Bookmark) = 0 Then
```

```
    SSDBGrid1.TagVariant = "<FONT COLOR=" & Chr$(34) & _
    "#FF0000" & Chr$(34) & ">This item is out of stock. " & _
    "Please try again tomorrow.</FONT><BR>"
Else
    SSDBGrid1.TagVariant = ""
End If
```

This code is similar to what you entered in the **RowLoaded** event. Depending on the quantity of an item, it assigns a value to the **TagVariant** property of the grid. Since the Data Grid will replace the {SSREPLACE TAGVAR} token with the value of the **TagVariant** property, this provides a handy way to insert a custom HTML snippet into the generated file "on the fly. "

9. Save the HTML file, run your project and click the "Export" button. This time, because you have specified the ssExportAllRows value for the *Flags* parameter, a new HTML file will be generated for each row in the grid, regardless of whether or not it is selected. If you open the `ITEM4.HTM` file in your web browser, you will see the following:



The control has replaced the {SSREPLACE TAGVAR} token with the contents of the **TagVariant** property, which inserts the relevant HTML into the generated file. If you open any of the other files, they will not contain the text in red, because they were generated from data that did not meet the condition specified in **RowExport**.

You have now seen how the row export feature can be used to generate individual HTML files for rows in the grid. You have also seen the further use of {SSREPLACE} tokens to "fill in the blanks" with grid data without using a generated table. In the next exercise, you will combine these techniques to create a master/detail form .

DBGrid: Ex. 7, Part IV - Exporting HTML

You have now seen how to use the **Export** method, **RowExport** event and HTML templates to create several types of customized HTML pages using the data and formatting from your Data Grid control. In this final exercise, you will see an additional feature of Data Widgets 3.1's exporting capabilities - the ability to create and embed hyperlinks in an exported document.

This exercise will build on the previous ones. The main table will be enhanced to provide hyperlinks to the detail page for each product. In addition, each product page will have a table detailing the items that are in stock. The information for this table will be drawn from the second grid and appended to the existing item pages.

1. If you do not have your project from the previous exercises available, open it now. Add a new form ("Form2") and add a Data Grid and a data control to the second form. Name the grid "SSDBGrid2" and the data control "Data2.

2. Specify the SMALL.MDB file as the **DatabaseName** property of the data control. Set the **RecordSource** property to "Items." Set the **DataSource** property of the data grid to "Data2."
3. Right-click on the grid and choose "Properties..." from the context menu. The property pages for the grid will appear. Click the "Load..." button and enter the location of the file ITEMS.GRD. (You may want to click the "..." button and browse for the file. It is located in the SAMPLES/CHAP05/HTML subdirectory of the directory where you installed Data Widgets.) This is a saved layout file for the fields in the Items database.
4. Add the following code to the **Load** event of Form1:

```
Form2.Show
```

Then add the following code to the **Reposition** event of Data1:

```
If SSDBGrid1.Columns(0).Text <> "" Then
   Form2.Data2.RecordSource = "SELECT * FROM Items WHERE [ItemID]=" & _
   SSDBGrid1.Columns(0).Text

   Form2.Data2.Refresh
   Form2.SSDBGrid2.Refresh
End If
```

This code changes the record source of the second grid to consist of only the items that have an ItemID value that matches the row selected in the first grid.

5. Run your project and click on the rows in the first grid (SSDBGrid1) one at a time. As you select an item, you should see the contents of the second grid change to reflect the detail information for that item. For instance, if you choose an item that has three units in stock, three rows will appear in the second grid with more information about those units.
6. Now that you have set up your data sources, it is time to modify the HTML templates to include the new features. Begin by opening the STOCKTPL.HTM template file in Notepad or your pure HTML editor. Save the file with a new name - MAINTPL.HTM. The existing code (from STOCKTPL.HTM) should look like this:

```
<HTML>
<BODY>
<CENTER>{SSREPLACE TABLE DATA B I COLOR CAPTION COLHEAD
ALIGN BGCOLOR WIDTH COLSPAN}</CENTER>
</BODY>
</HTML>
```

Replace the existing code with the following, then save the file using the new name:

```
<HTML>
<BODY LINK="#00FFFF" ALINK="#00FF00" VLINK="#FFFF00">
<CENTER>
{SSREPLACE DATA TABLE B I COLOR CAPTION COLHEAD
ALIGN BGCOLOR WIDTH COLSPAN LINKANCHOR="Description"
LINKMASK="ITEM.HTM" LINKFIELD="ItemID"}
</CENTER>
</BODY>
</HTML>
```

The new code is just like the code you replaced, except for a few additions. Some attributes have been added to the <BODY> tag to ensure that the color of the link text is visible against the background colors of the grid. But more importantly, three new attributes have been added to the {SSREPLACE} token. The LINKANCHOR attribute determines which field in the table will be made into a hyperlink. When the HTML is generated, the "Description" field will be enclosed by <A HREF=> tags that point to another HTML file. The file pointed to is determined by the values of LINKMASK and LINKFIELD. These attributes are similar to the *ExportToFile* and *OutputFileField* parameters of the **Export** method. The value of the field specified

by LINKFIELD is appended to the filename specified by LINKMASK to create the full name of the file that is the target of the link.

7. To create the detail pages, you will need a template for the data from the first grid and a different template for the data from the second grid. Items from SSDBGrid1 will be exported as individual row files, just as they are now. Detail information for each item (from SSDBGrid2) will be exported as an HTML table. During the export, the detail tables will be appended to the row file pages, creating a single HTML page for each row in SSDBGrid1.

   You will actually create two templates that will be used to generate the second half of the HTML page; one that will display information from the second grid (inventory for items that are in stock) and one that will display a message if there is no information (for items that are out of stock.)

   Open the `1-ROWTPL.HTM` file in your editor and save it with the name `PART1TPL.HTM`. Replace the contents of the file with the HTML code below. (This code is based on the code in `1-ROWTPL.HTM`.)

```
<HTML>
<BODY>
<B>ITEM NUMBER {SSREPLACE DATA FIELD="ItemID"}</B><BR>
<FONT SIZE=5 COLOR="#0000FF"><B>
{SSREPLACE DATA FIELD="Description"}</B></FONT>
<HR>
Price: {SSREPLACE DATA FIELD="Price"}<BR>
Quantity in Stock: {SSREPLACE DATA FIELD="StockQty"}<BR>
<BR>
<I>Preferred Customer Discount: {SSREPLACE DATA FIELD="Discount"}</I>
<P>
```

   Because this is only the template for the first half of the page, it is an incomplete HTML file. It will be complete when the second part is appended. <FONT> tags have been added to the name of the item to make it stand out more, now that the page contains more information. Note that the {SSREPLACE TAGVAR} token has been removed from the code.

8. Create a new HTML file called `PART2TPL.HTM` and enter the following code in it:

```
{SSREPLACE DATA TABLE B I COLHEAD ALIGN BGCOLOR WIDTH COLSPAN}
<P>
To order call: 1-800-555-SMALL<P>
<P>
<A HREF="INSTOCK.HTM">Back to product list</A>
</BODY>
</HTML>
```

   This code adds the detail table and completes the HTML page. It also includes a link back to the main table page, to make navigation easier.

9. Save the `PART2TPL.HTM` file with a new name, calling it `PARTXTPL.HTM`. This is the template file that will be used in the event an item is out of stock. The only difference between `PART2TPL.HTM` and `PARTXTPL.HTM` is that the {SSREPLACE} token has been replaced by a fixed line of HTML. Enter the following code in the new file, then save it.

```
<FONT SIZE=4 COLOR="#FF0000">This item is out of stock.
Please try again tomorrow.</FONT><BR>
<P>
To order call: 1-800-555-SMALL<P>
<P>
<A HREF="INSTOCK.HTM">Back to product list</A>
</BODY>
</HTML>
```

10. Finally, the code used to export the grid data will have to be rewritten. The following code is based on the

existing code, but interacts more closely with the two Data Grids to produce the correct HTML.

First, delete all the code from the **RowExport** event of SSDBGrid1. Since the program now uses a separate template for items that are out of stock, the "out of stock" message no longer needs to be inserted during generation.

Then add the following code to the **RowExport** event of SSDBGrid2:

```
'Replace true and false values with text
If SSDBGrid2.Columns(5).Text = "0" Then
   SSDBGrid2.Columns(5).Text = "No"
Else
   SSDBGrid2.Columns(5).Text = "Yes"
End If
```

This code examines the contents of the Boolean column in the second Data Grid, and replaces the raw values with their English equivalents.

Delete all the code in the **Click** event of the "Export" button and replace it with the following code. Code comments are optional:

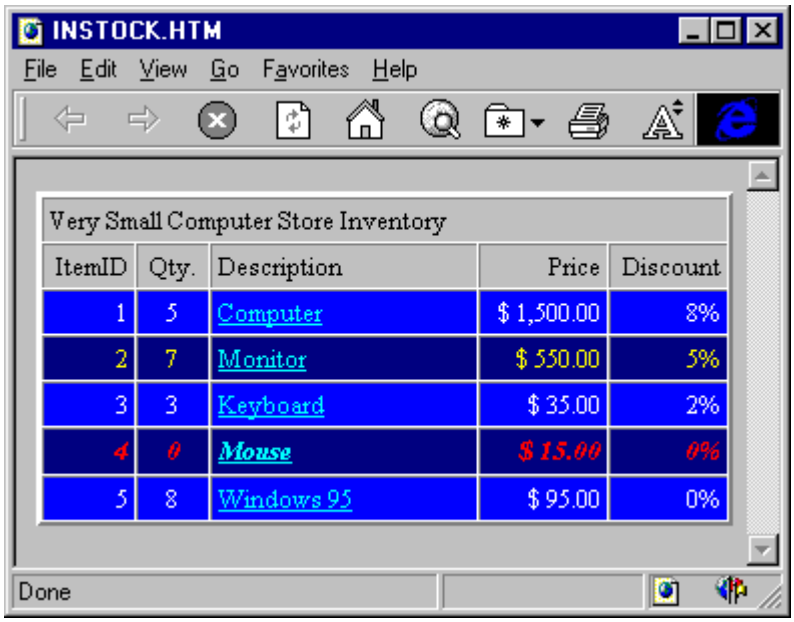
```
Dim sTemplateName As String

'Export master file
SSDBGrid1.Export ssExportTypeHTMLTable, ssExportAllRows, _
Or ssExportOverwriteExisting, "INSTOCK.HTM", "MAINTPL.HTM"

'Export first part of item files
SSDBGrid1.Export ssExportTypeHTMLRowFiles, ssExportAllRows _
Or ssExportOverwriteExisting, "ITEM.HTM", "PART1TPL.HTM", _
"ItemId"

Data1.Recordset.MoveFirst
SSDBGrid1.Refresh

Do While Not Data1.Recordset.EOF
   Form2.Data2.Refresh
   Form2.SSDBGrid2.Refresh

   'Determine whether there are any rows in the item table
   'and use the appropriate template
   If SSDBGrid1.Columns(1).Text = "0" Then
      sTemplateName = "\PARTXTPL.HTM"
   Else
      sTemplateName = "\PART2TPL.HTM"
   End If

   'Export second part of item files
   Form2.SSDBGrid2.Export ssExportTypeHTMLTable, _
   ssExportAllRows Or ssExportAppendToExisting, "ITEM" & _
   SSDBGrid1.Columns(0).Text & ".HTM", sTemplateName

   Data1.Recordset.MoveNext
   SSDBGrid1.Refresh
Loop
```

The first part of the code is similar to the original code of the **Click** event. The **Export** method is used to create a table containing all the rows from SSDBGrid1, but this time using the new `MAINTPL.HTM` template. Then the row files are created through a second call to the export method. This is the same as in Part III of the exercise, except that the `PART1TPL.HTM` file is being used to create the files. The result is a file for each row in the grid that is the first half of a completed web page.

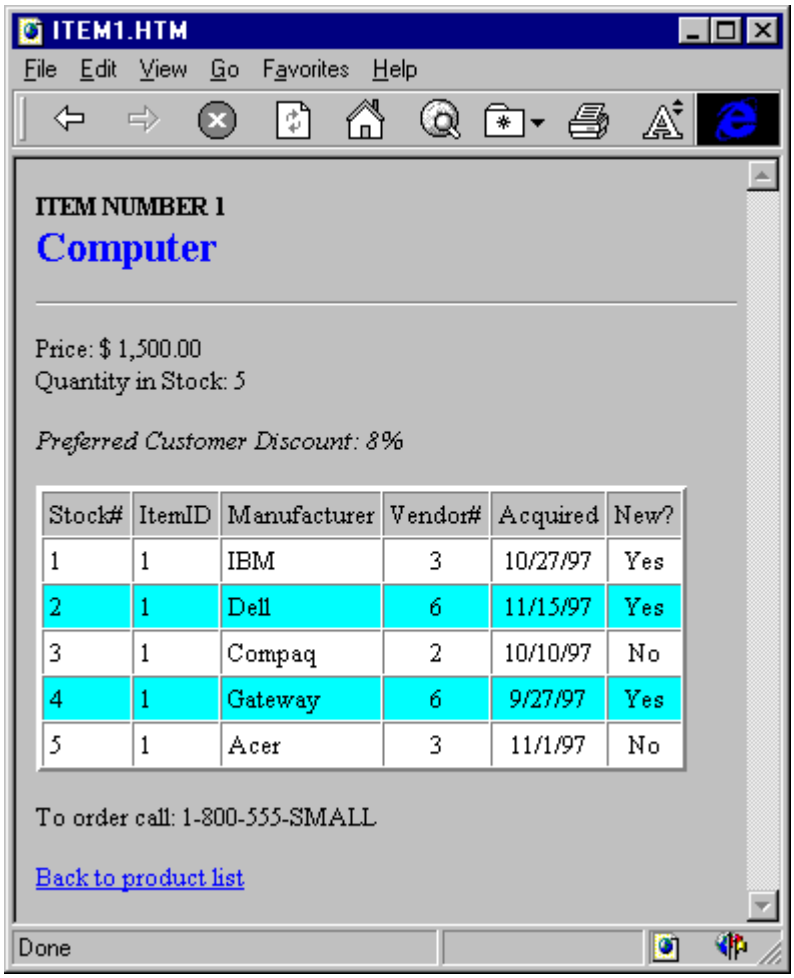
The new functionality begins when SSDBGrid1 is reset so that the record pointer is on the first record in the database. The `Do... Loop` structure that follows steps through each record in SSDBGrid1. SSDBGrid2 is populated based on the record selected in SSDBGrid1. As each record in SSDBGrid1 is selected, it is examined to see whether the Quantity field is equal to zero. This determines which template will be used to export the data from SSDBGrid2.

Then the **Export** method of SSDBGrid2 is invoked. The filename for the output file is created based on the ItemID value from SSDBGrid1, which ensures that the filename will correspond to one of the files that was already generated. The name of the template file has already been determined. However, instead of overwriting the existing files, this **Export** method appends its output to them (*ssExportAppendToExisting*). The result is that each incomplete row file has either a detail table or an "out of stock" message appended to it, at the same time becoming a complete HTML file.

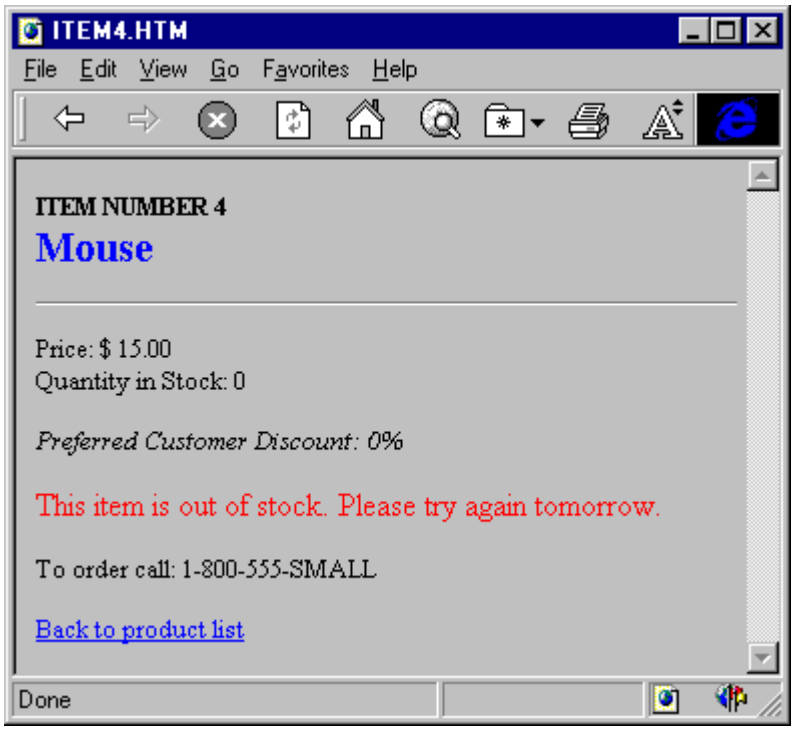
11. Run the project and click on the "Export" button. As before, an `INSTOCK.HTM` file will be generated, along with a number of `ITEM?.HTM` files. Open the `INSTOCK.HTM` file in your web browser, and you will see that the item descriptions are now hyperlinks:



Click on the item description for the first item (Computer). You will jump to the detail page that lists the price of the item and the details for each item of that type that is in stock:



Click the link at the bottom of the detail page to return to the master page, then click on the description for Item #4 (Mouse). This time you will see a different detail page that was generated for the out-of-stock item:



This concludes the Data Widgets 3.1 HTML export tutorial. You have seen how to export grid data into HTML format in a variety of ways; with and without formatting, including or excluding various grid attributes, by generating table and row files or even by combining the two. To find out more, consult the sample projects in the `WEBNAV` subdirectory of your `SAMPLES` directory.

Data Combo Guided Tour

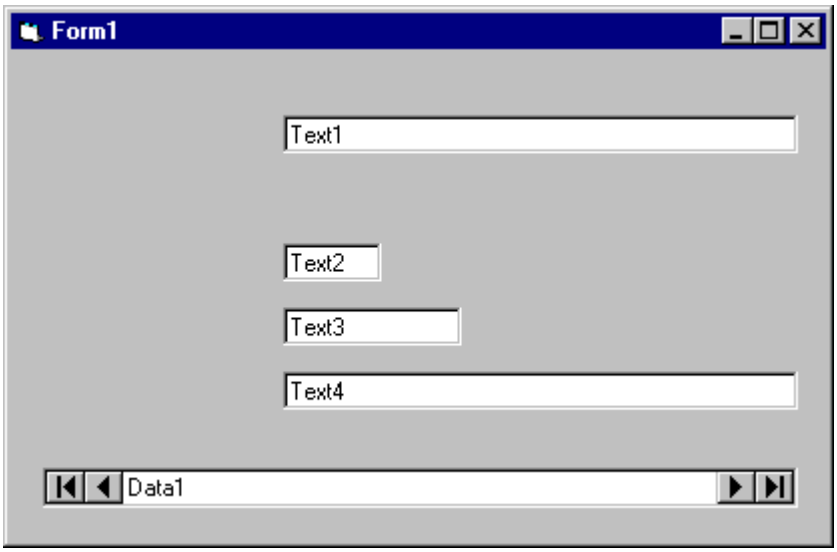
Data Combo: Exercise 1

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

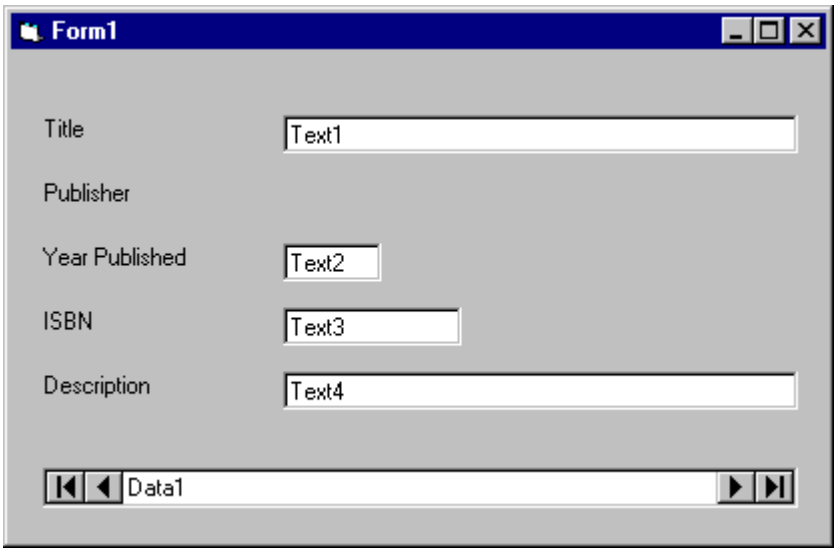
For the exercises in this section, it is assumed that you have already launched your development application (Visual Basic), and that the control has been added to your toolbox. For more information on how to do this, refer to Using Data Widgets.

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:



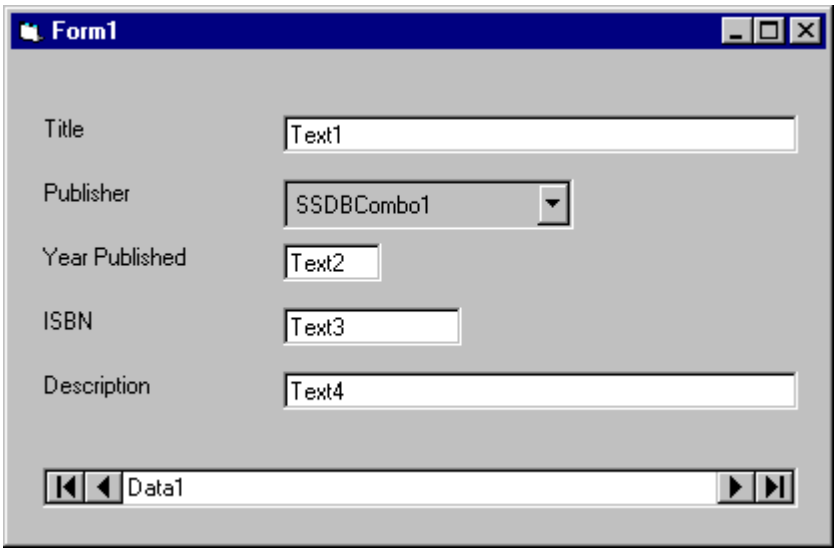
5. Create five text labels on the form. Your form should look like:



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:



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 DropDown Guided Tour

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).DropDownhWnd = 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.

Enhanced Data Control Guided Tour

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.

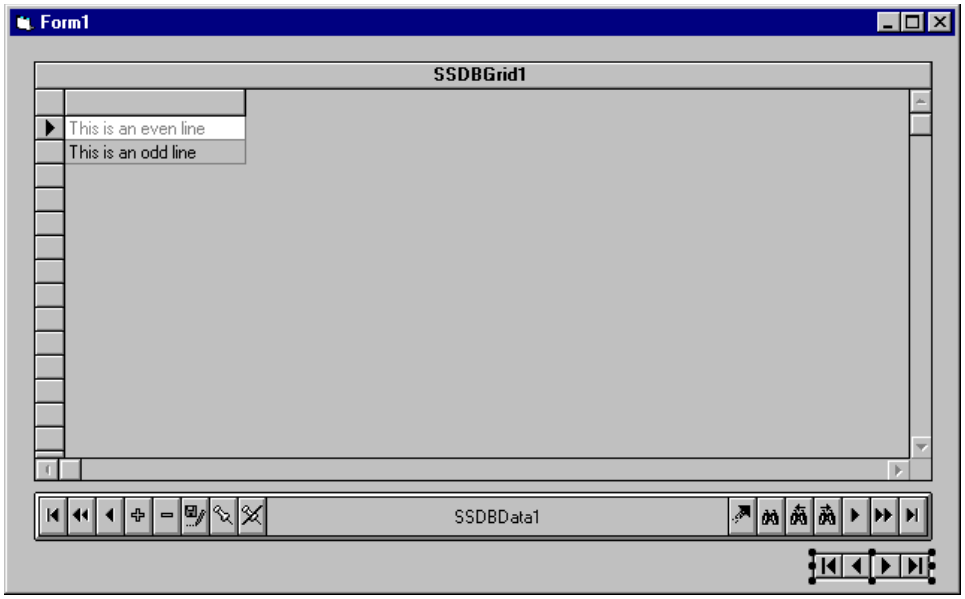
You will create an application that makes use of the Enhanced Data control. The database used will be the BIBLIO.MDB that ships with Visual Basic and is located in the Visual Basic home directory. If you are using an environment other than Visual Basic, and do not have the BIBLIO.MDB database, consult 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.

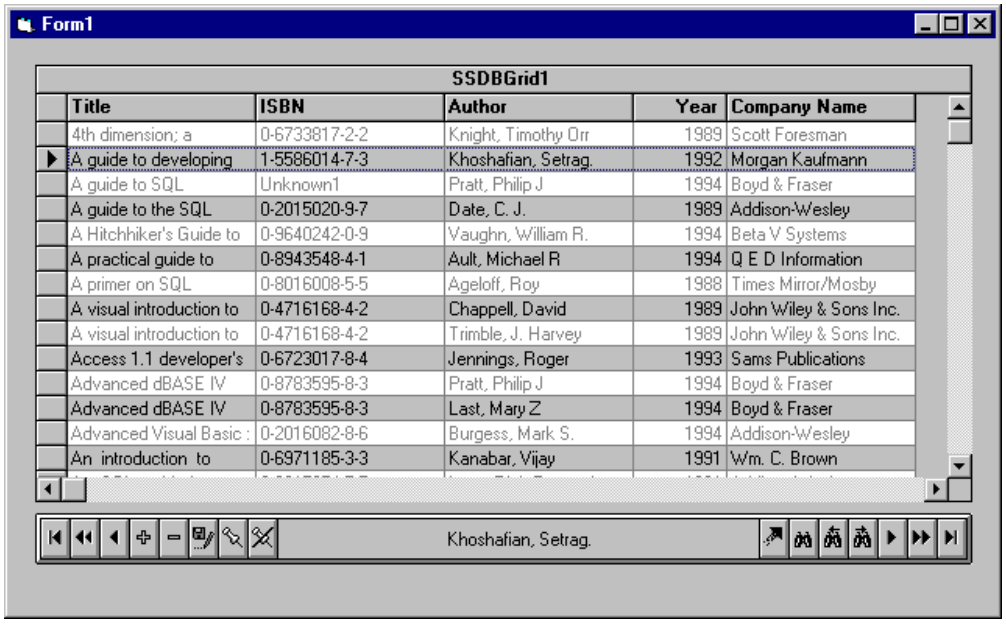
   Resize the grid to a size that is suitable for your form.

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.

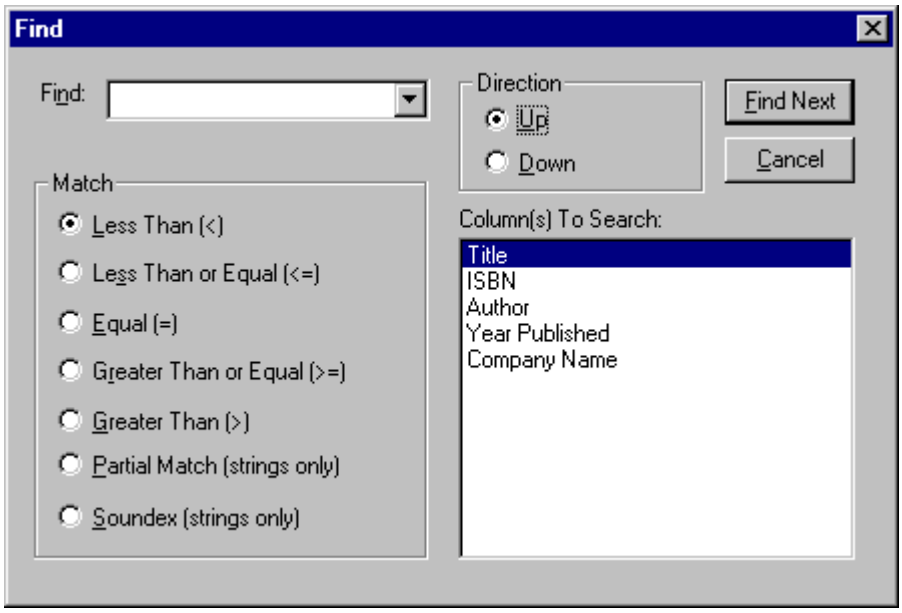
Update Record
Updates the selected record in the database.

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 though 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 though 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.

Data OptionSet Guided Tour

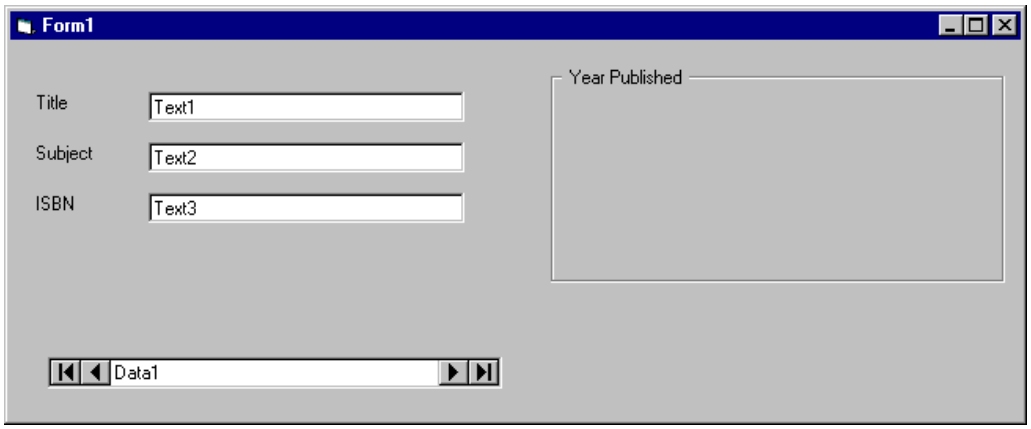
DataOptionSet: Exercise 1

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.

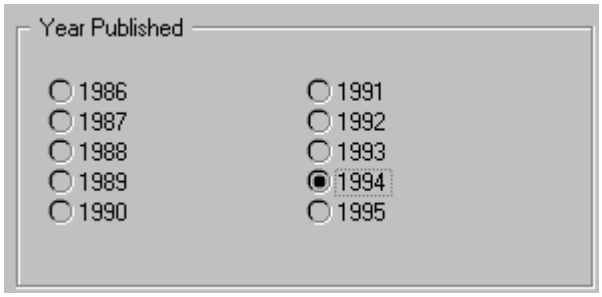
In this exercise, you will create an application that makes use of the DataOptionSet control. The database used will be the BIBLIO.MDB that ships with Visual Basic and is located in the Visual Basic home directory. If you are using an environment other than Visual Basic, and do not have the BIBLIO.MDB 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'.



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:



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.

Data Command Button Guided Tour

Data Command: Exercise 1

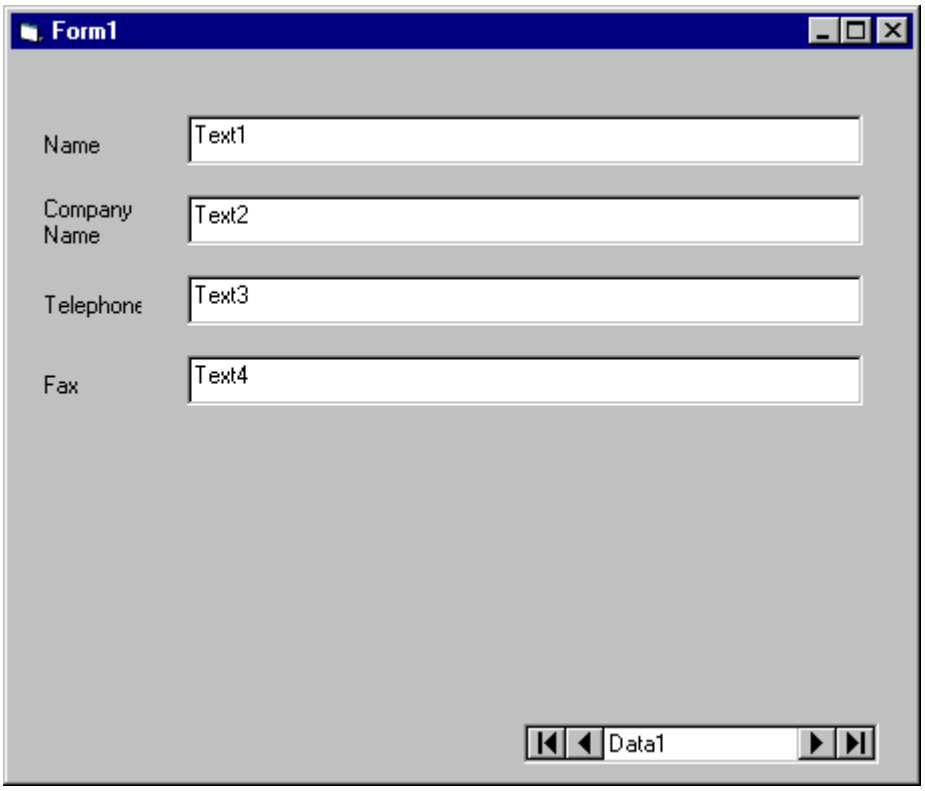
This guides you through the creation of a sample program using the Data Command control. For a complete description of this control, refer to the Data Command section.

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'.



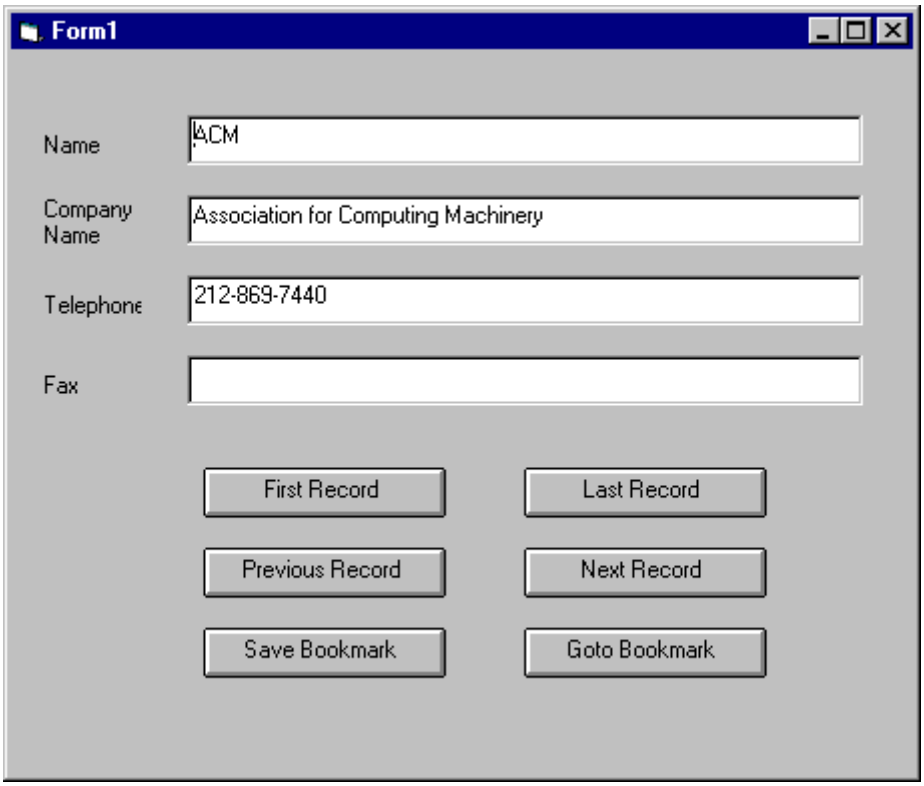
5. Add six Data Command buttons to the form setting the **DataSource** property for each one to 'Data1' and changing the **Caption** property for each one so that they look like this:

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.



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.

Included Samples

Visual Basic

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 defualt by using only the code in theForm_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 intial 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
```

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 iTotal 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(iC)
      iTotal = iTotal + SSDBGrid1.Columns(0).CellValue(vBM)
   Next iC

   Text1.Text = iTotal

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.

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 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 readablity 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 diplaying 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.

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 or 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, whcih 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
```

Visual C++

Important Note on Using VC++ Examples

If you are using Data Widgets 3.1 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 it's `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.

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();
```

VC++ Adding Columns at Run Time example

Note Assume that *m_grid* is an object of CSSDBGrid. The code snippet below shows how to reference the columns collection, add two columns, reference a particular column object in the collection and change the **Caption** of each column.

```
// create the columns
CSSColumns cols = m_grid.GetColumns();
cols.RemoveAll();
CSSColumn col;

// add Author ID column
cols.Add(0);
col = cols.GetItem(COleVariant(short(0), VT_I2));
col.SetCaption(_T("Author ID"));

// add the Author field column
```

```
cols.Add(1);
col = cols.GetItem(COleVariant(short(1), VT_I2));
col.SetCaption(_T("Author"));
```

You can access the Groups collection and Group objects in a similar manner to the Columns and Column objects.

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
CSSDBGrid* pGrid = (CSSDBGrid*)GetDlgItem(IDC_SSDBGRIDCTRL1);
mygrid.Create(pGrid);

ISSSelBookmarks* 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:
//
```

VC++ ISSPrintInfo Example

Note This example assumes that *CTestDlg* is a Class Wizard generated file and *OnPrintBeginSsdbgrid1* is the name of the **PrintBegin** event handler.

```
void CTestDlg::OnPrintBeginSsdbgrid1(LPDISPATCH ssPrintInfo)
{
   // TODO: Add your control notification handler code here
   ISSPrintInfo printInfo(ssPrintInfo);

   // print 3 copies and colors
   printInfo.SetCopies(3);
   printInfo.SetPrintColors(TRUE);
}
```

VC++ ISSRowBuffer Example

Note This example assumes that *CTestDlg* is a Class Wizard generated file based and *OnUnboundReadDataSsdbgrid1* is the name of the **UnboundReadData** event handler.

```
void CTestDlg::OnUnboundReadDataSsdbgrid1(LPDISPATCH RowBuf, VARIANT
FAR* StartLocation, BOOL ReadPriorRows)
{
   // TODO: Add your control notification handler code here
   ISSRowBuffer rBuf(RowBuf);
   CString s;
   s.Format("Number of rows to read = %ld", rBuf.GetRowCount());
   AfxMessageBox(s);

// Add code here to read the data from an array or recordset
// object into the RowBuffer object using the SetValue() method
// e.g.
// CString sMyText = _T("This is my text");
// rBuf.SetValue(0, 0, COleVariant(sMyText));
}
```

VC++ Optional Parameter Helper Class

```
// EmptyOleVar.h - use this class for passing optional parameters
```

```
// in VC++ (this emulates VB's optional parameters functionality)

class CEmptyOleVar : public COleVariant
{
public:
   CEmptyOleVar() {
      this->vt = VT_ERROR;
      this->scode = DISP_E_PARAMNOTFOUND;
   };
};
```

Inserting controls into your VC++ project

The method you use to insert the Data Widgets controls into your Visual C++ project will depend on which version of Visual C++ you are using. Also, there are special concerns you should be aware of relating to the use of variant values with the Data Widgets controls. Refer to the following sections for more information:

- **Visual C++ 5.0**
- **Visual C++ 4.2**
- **Generated Files**
- **Working With Variants**

Visual C++ 5.0

To use Data Widgets controls in a VC++ 5.0 project, you must generate a corresponding set of wrapper classes and include them in your project. You only have to generate the classes once for a given project. Wrapper classes are reusable, so once you have created them you can use them in other projects that make use of the Data Widgets controls.

Take the following steps to create the wrapper classes for a Data Widgets control in your Visual C++ 5.0 project:

1. Use the Resource Editor to insert tht Data Widgets control you want to use into a dialog box template.
2. From the Resource Editor, invoke the Class Wizard (Ctrl-W).
3. Click on the "Add Class..." drop-down button and select "From a Type Library..."
4. From the dialog box which comes up, choose the ISSPrintInfo and ISSRowBuffer classes and click on the OK button. This will generate the **ssdw3b32.h** and **ssdw3b32.cpp** source files and include them in your project.
5. If necessary, use Class Wizard to create a class for your dialog resource and then add a variable pertaining to the appropriate Data Widgets control under the **Member Variables** tab in Class Wizard.
6. A dialog box should present you with the names of the wrapper classes which will be generated. Each of these classes contain Get and Set methods for accessing the control's properties and object collections.You can then use these classes in the same manner that you would use any other C++ classes. Be sure to include the appropriate header files as necessary.

Visual C++ 4.2

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 SSDW3B32.OCX. When you select this file, ClassWizard will retrieve the necessary wrapper classes for use in your project.

Generated Files

ssactivecell.h and **ssactivecell.cpp -** contains wrapper class for ActiveCell object

sscolumn.h and **sscolumn.cpp -** contains wrapper class for Column object

sscolumns.h and **sscolumns.cpp -** contains wrapper class for the Columns collection object

ssdbgrid.g and **ssdbgrid.cpp -** contains wrapper class for SSDBGrid Object

ssdw3b32.h and **ssdw3b32.cpp -** contains wrapper class for PrintInfo and RowBuffer interfaces

ssgroup.h and **ssgroup.cpp -** contains wrapper class for Group object

ssgroups.h and **ssgroups.cpp -** contains wrapper class for Groups collection object

ssselbookmarks.h and **ssselbookmarks.cpp -** contains wrapper class for the SelBookmarks collection

ssstyleset.h and **ssstyleset.cpp -** contains wrapper class for the StyleSet object

ssstylesets.h and **ssstylesets.cpp -** contains wrapper class for the StyleSets collection object

Working With Variants

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());
```

Using Data Widgets

Data Grid

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 an Unbound Data Grid to your application

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 AddItem Grid to Your Application

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

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 crid, 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.

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 baesd 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
```

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 conatins 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.

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.

Note There is now a Guided Tour that covers this subject in depth, with complete explanations of how the code works. Click here to jump directly to that tutorial.

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 decalred 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.

   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.
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.

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

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 intial 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.

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.

How the Data Grid Handles Data Validation and Error Checking

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

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 sting ("") in the database.

Note that this applies only to String type fields.

Transferring a Layout Between Grids with Different Data Sources

There are two ways to save a grid layout in Data Widgets 3.1 - one saves only the StyleSets defined in the current grid, while the other saves a full layout that includes a wide array of configuration information. (For a complete list of the attributes saved in a grid layout file, click here.) The StyleSets Only option makes it easy to transfer StyleSets among any grid-like controls, regardless of their underlying data source.

The Full Layout method can be used to transfer advanced layout options between controls with different data sources. However, the process is not completely automatic, as it is with StyleSets. Because data binding information is saved as part of a layout file, there are some concerns you must address to use this approach.

When you apply a stored layout to a control from a layout file, the existing layout of the grid is destroyed. Then the stored columns are added to the grid with formatting and data binding information intact. If there are any field names in the stored layout that match the field names in the current data source, those columns will be connected to the appropriate fields. Columns bound to a data field that does not match one of the fields from the current data source will become unbound columns.

Once the columns from the layout have been restored, you can manually step through the Columns collection and bind any unbound column to a field in the data source. You can also change the text of the column header or any other formatting, as necessary.

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.

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.

Export The Data From a Data Grid to HTML

There are two basic steps to exporting HTML. First, you must create a *template file* that will be used to format the exported data. A template file is simply an HTML file that contains one or more *replacement tokens*. When the control exports an HTML file, it uses the structure of the template but replaces the replacement tokens with data from the grid. The second step is to invoke the **Export** method with the proper parameters to generate the file.

1. Using a plain text or pure HTML editor, create the following HTML file. All line breaks are optional.

```
<HTML>
<BODY>
{SSREPLACE TABLE DATA CAPTION COLHEAD GRPHEAD
FACE B U EM STRIKE SIZE COLOR BGCOLOR ALIGN
WIDTH LEVELS}
</BODY>
</HTML>
```

2. Save the HTML file with a name of your choice. Remember the name and location of the file, as you will need it soon. For this example, assume the name of the file is C:\SHERIDAN\TEMPLATE.HTM.
3. Add a button or menu option to your application to initiate the export. You may also initiate the export in response to some other program event.
4. Add the following code to the event that initiates the export. This code assumes you used the example name specified in step 2. If you used another file name, substitute it in place of C:\SHERIDAN\TEMPLATE.HTM. Also assumed is the name of your grid control; substitute the actual name of your control for SSDBGRID1.

```
SSDBGrid1.Export ssExportTypeHTMLTable, _
ssExportAllRows OR ssExportOverwriteExisting, _
"C:\SHERIDAN\EXPORTED.HTM", _
"C:\SHERIDAN\TEMPLATE.HTM"
```

Using this template and this code will create an HTML page called C:\SHERIDAN\EXPORTED.HTM that will contain all the data from the Data Grid and any grid formatting that can be translated into HTML (colors, font size, attributes such as boldface & italics, alignment, etc.). If a file with the same name already exists, that file will be overwritten by the new file. If you wish to give the generated file a different name, substitute that name for C:\SHERIDAN\EXPORTED.HTM in the **Export** method.

You can export data to HTML in many different ways. This procedure offers you the simplest and most straightforward method of exporting data. Data export is controlled by both the parameters passed to the **Export** method and the attributes specified for the replacement tokens in the HTML template file. To learn more about exporting grid data to HTML, review the HTML Export Tutorial.

Setting Up A Print Job In The PrintInitialize Event

When you invoke the **PrintData** method, the DataGrid generates an **ssPrintInfo object** and immediately passes it to the **PrintInitialize** event. This object contains default formatting information about the pending print job. You then change the values of the ssPrintInfo object's properties to control the formatting of the print job.

For example, suppose you wanted three collated copies of a report, in landscape mode, with half-inch margins on each side. You would add the following code to the **PrintInitialize** event:

```
With ssPrintInfo
   .Collate = True
   .Copies = 3
   .Portrait = False
   .MarginLeft = 0.5
   .MarginRight = 0.5
   .MarginTop = 0.5
   .MarginBottom = 0.5
End With
```

When invoking the **PrintData** method, you can specify whether or not to display Print Setup and Print dialogs to the user. If you choose to do so, the values you assign to the ssPrintInfo object in the **PrintInitialize** event will be used to fill in the values that will be displayed to the user in the dialog boxes. The user can then change any of the values you have specified.

To find out the values the user specified in the dialogs, you would use the **PrintBegin** event.

Using Stored Layouts To Print Pre-Designed Reports

By taking advantage of Data Widgets 3.1's capability to save and restore grid layouts, you can provide the user of your program with pre-defined report layouts. Any gird attribute that can be stored in a layout can be applied to a printed report, including column and group arrangements, fonts, colors and StyleSets (including pictures.)

To create a prepared report.

1. Create a Data Grid that contains the data that will be printed in the report.
2. Set up the grid layout the way you want it to appear in the report. You can do this at design time by using the Grid Editor on the Data Widgets property pages, or through code.
3. Save the layout you have created in a layout file. Depending on how you created the custom layout in step 2, take one of the following actions:
   - If you created the layout at design time using the Grid Editor, click the "Save..." button at the bottom of the dialog and specify a file name for the layout file.
   - If you created the layout through code, invoke the **SaveLayout** method (using the *ssSaveLayoutAll* flag) with a filename for the report layout.
4. Create the code that will enable the user to print using a custom report format. This might entail adding a menu option that lists the available reports, or displaying a File Open dialog to let them choose a grid layout (.GRD) file. You can also hard-code which template will be used. Whichever way you choose, your program should know which report file will be used before you invoke the **PrintData** method.

5. In the event triggered by the user to print the report, take the following steps in the order shown:

   a.) Set the **Redraw** property of the grid to False.

   b.) Use the **SaveLayout** method of the grid to save the existing grid layout to a temporary file.

   c.) Use the **LoadLayout** method of the grid to load the selected layout (.GRD) file. This will apply the report formatting to the current grid.

   d.) *Optional:* If you want to specify font formatting for the report headers or footers, set the appropriate values for the **PageHeaderFont** and **PageFooterFont** properties of the grid.

   e.) Invoke the **PrintData** method to initiate the printing of the report using the new layout.

   f.) Use the **LoadLayout** method to restore the original grid layout from the temporary file you created in step 5b.

   h.) Set the **Redraw** property of the grid to True.

6. In the **PrintBegin** event of the grid, do the following:

   - *Optional:* Specify any special header or footer text, using the PageHeader and PageFooter properties of the ssPrintInfo object which is passed to the event.

Note The layout (.GRD) files you create must be distributed along with your application.

Using The RowPrint Event To Examine And Change Report Data

While a report is being created, the **RowPrint** event will occur once for each row of data that is included in the report. Because the bookmark of the row data is available in the event, it is possible to examine the contents of any cell and take action based on what you find. You can also change or reformat row contents "on the fly" - changes will be reflected in the printed report, but not in the displayed grid.

To see how this works, suppose you have a Data Grid that stores some confidential information in column 8. The level of security is stored in column 3, and determines whether the sensitive information in that row may appear in a printed report. You would add the following code to the **RowPrint** event:

```
iAccessLevel = SSDBGrid1.Columns(3).CellValue(Bookmark)
'Data may only appear in report if
'access level is greater than five.
If iAccessLevel <= 5 Then
   'Block out sensitive data in column 8
   SSDBGrid1.Columns(8).CellText(Bookmark) = _
   "XXX Confidential XXX"
End If
```

Using The PrintBegin Event To Examine User Settings

When you invoke the **PrintData** method to create a report of the data in the grid, you must specify whether or not to display Print Setup and Print dialogs to the user. The default values displayed by these dialogs are the ones assigned to the **ssPrintInfo** object in the **PrintInitialize** event.

If the user makes any changes to the values in either of the two dialogs, those changes will be reflected in the **ssPrintInfo** object when it is passed to the **PrintBegin** event. In this event, you can examine the changes the user has made and take appropriate action. You may want to change the choices the user has made, or store their preferences for use in subsequent print jobs.

For example, suppose you want to save the margins the user has selected to a variable called udtUserMargins, which is a user-defined type variable you have created to store printer information. Also, to avoid excessive printer traffic, you want to make sure that the user prints no more than ten copies of the report at one time. You would use the **PrintBegin** event to implement both of these features by adding the following code:

```
udtUserMargins.Left = ssPrintInfo.MarginLeft
udtUserMargins.Right = ssPrintInfo.MarginRight
udtUserMargins.Top = ssPrintInfo.MarginTop
udtUserMargins.Bottom = ssPrintInfo.MarginBottom

If ssPrintInfo.Copies > 10 Then
   ssPrintInfo.Copies = 10
   MsgBox "Only 10 copies may be printed at once."
End If
```

Data Combo

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 an Unbound Data Combo

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.

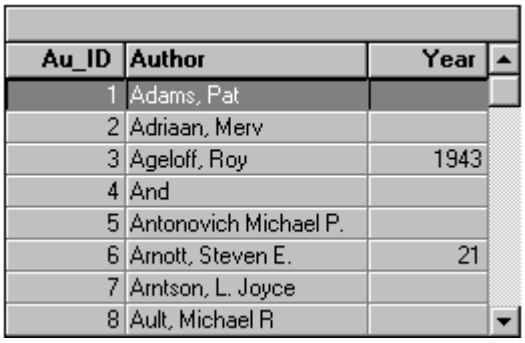
When the edit portion of the Data Combo is unbound, you need to initially supply the field value yourself via the **Text** property which contains the value of the data in the edit portion of the Data Combo. When the user clicks on the dropdown button, the list portion will automatically update the **Text** 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.

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	Arntson, L. Joyce	
8	Ault, Michael R	

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.

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.

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**.

Data DropDown

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 an Unbound Data DropDown

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.

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.

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.

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.

Enhanced Data Control

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.

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.

Data OptionSet

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 run time in order for the control to be useful.

Binding a DataOptionSet 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.

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".

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.

Data Command Button

Adding a Data Command Button

The Data Command button only performs database actions 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.

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 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 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.

Keyboard Interface

The following describes the keyboard interface for each of the Data Widgets controls that support keyboard use.

SSDBGrid

Keystroke	Action	Comments
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 deleted.
		AllowDelete must be set to **True.**
DEL	Deletes the selected row	In the case of multiple rows being selected, they will all be deleted.
		AllowDelete must be set to **True.**

SSDBCombo

Keystroke	Action	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 down, selects the current row and closes the dropdown.	Works only on the dropdown portion.

SSDBDropDown

Keystroke	Action	Comments
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

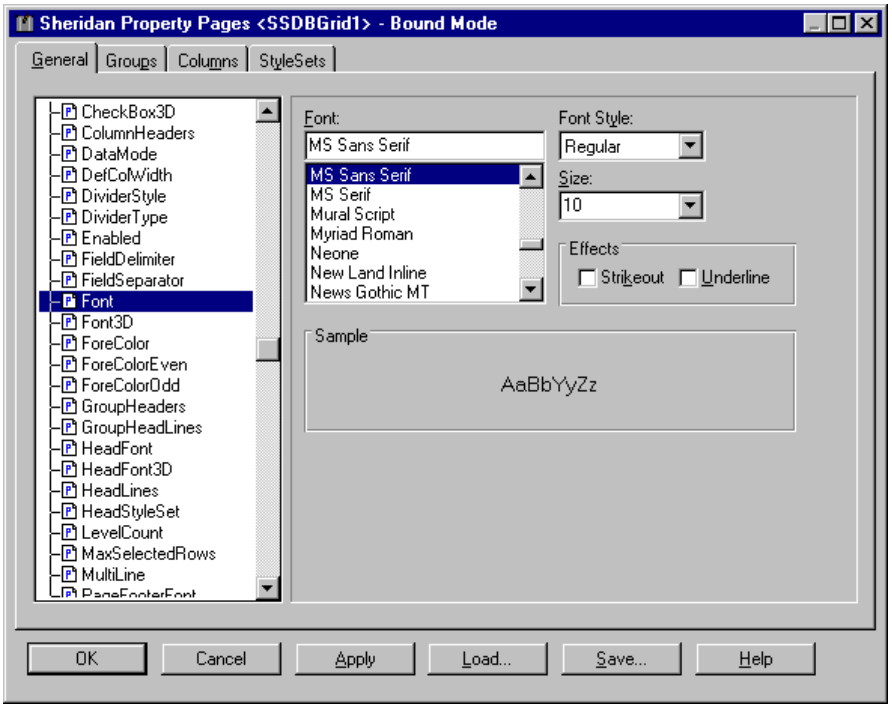
Keystroke	Action	Comments
UP ARROW	Moves up a button	
DOWN ARROW	Moves down a button	
LEFT ARROW	Moves up a button	
RIGHT ARROW	Moves down a button	
HOME	Moves to the first button in the set	
END	Moves to the last button in the set	

The 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.

You can also use the Grid Editor to load and save grid layout files that contain sets of formatting attributes and/or StyleSets. For more information on using this feature, see What's New / Save and Restore Grid Layouts & StyleSets



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: General tab

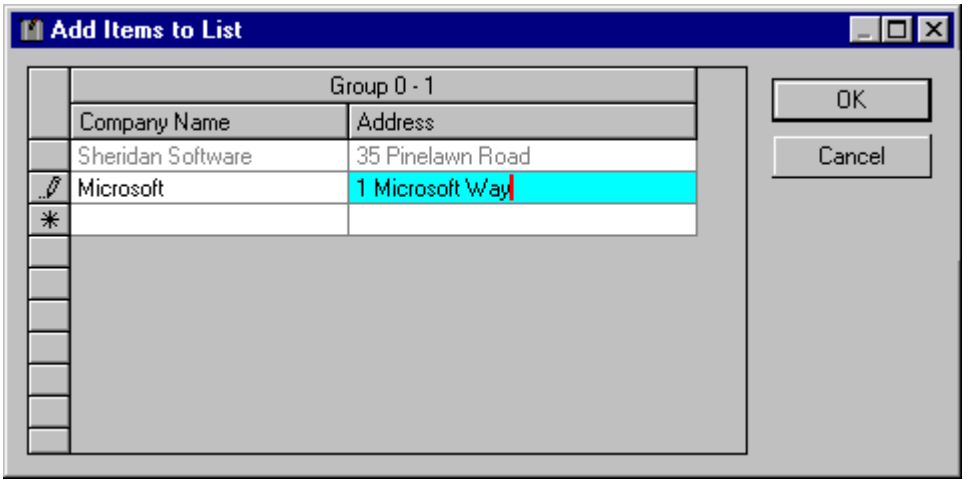
The General tab has the appearance of a standard Sheridan Property Page. Through a tree structure, you are able to select and modify properties that apply to the grid as a whole. To modify a property, simply select it from the tree and make the desired changes from the options presented on the right.

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:



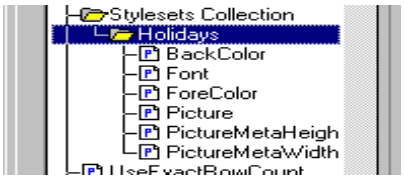
Fill in the data as needed, clicking the **OK** button when you are finished. Clicking the **Cancel** button allows you to exit without saving your changes.

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 interchangable 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:

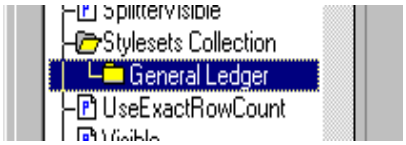
To see the individual properties applicable to the StyleSet, double-click that StyleSet:



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: Columns tab

The Columns Tab allows you to define the columns that apepar 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

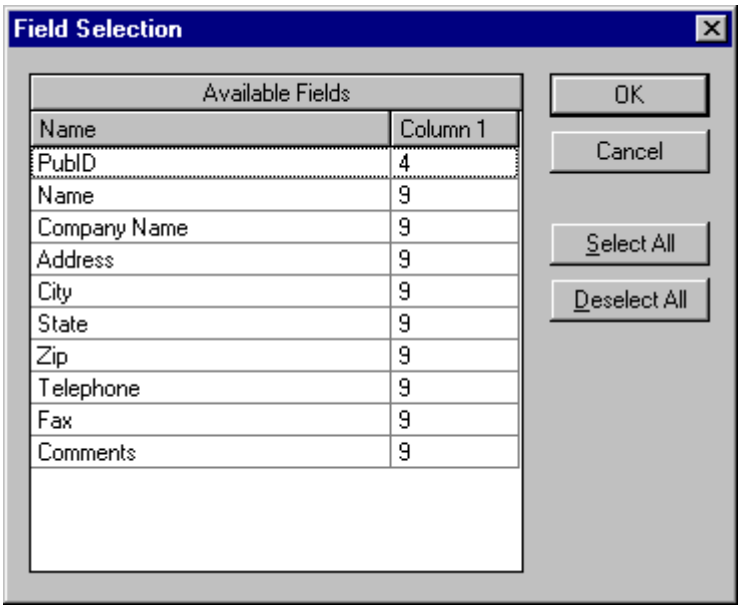
1. In the Design Grid, click the header of the column to remove.
2. Click the Remove button.

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: 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.

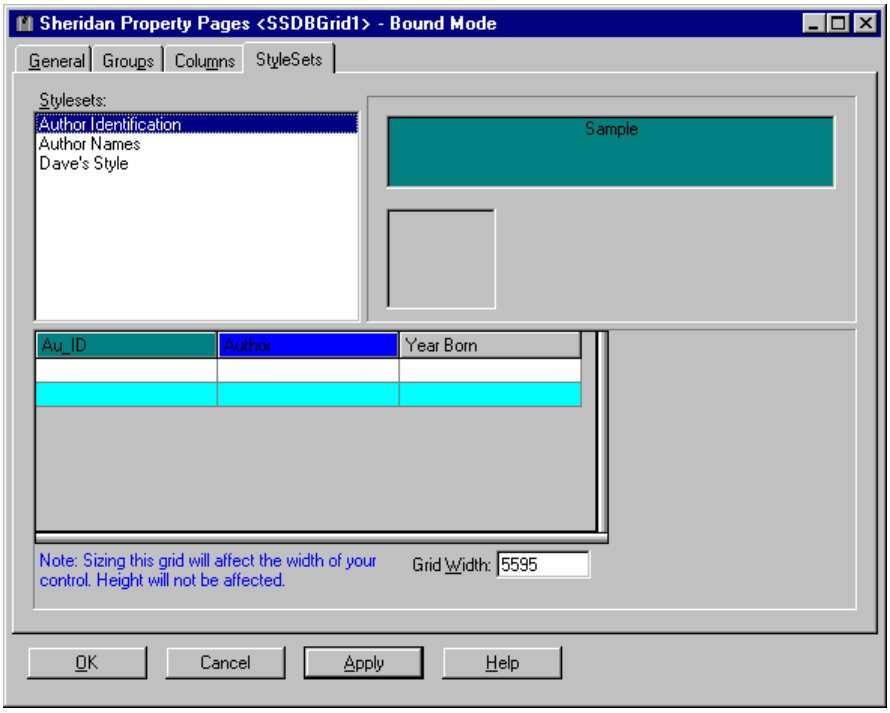
To set group specific properties:

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.



Data Widgets Control Reference

Properties, Methods, Events, Objects & Collections

(About) Property

Applies To

SSDBCombo, SSDBCommand, SSDBDropDown, SSDBData, SSDBGrid, SSDBOptSet

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

Applies To

SSDBCombo, SSDBCommand, SSDBDropDown, SSDBData, 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.

ActiveCell Method

Applies To

SSDBGrid

Description

Returns the active cell object.

Syntax

object . **ActiveCell**

Part	Description
object	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.

See Also

ActiveCell Object

ActiveCell Object

Description

An ActiveCell object represents the active cell in the grid.

Properties

Height	SelText	Value
Left	StyleSet	Width
SelLength	Text	
SelStart	Top	

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.

See Also

ActiveCell Method

ActiveRowStyleSet Property

Applies To

SSDBGrid

Description

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

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
text	A string expression that evaluates to the name of a **StyleSet** in the **StyleSets** collection, or an empty string ("").

Remarks

Specifying an emptry string removes the StyleSet applied to the currently active row.

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:

ActiveCell.StyleSet	(overrides all below)
Control. ActiveRowStyleSet	(overrides all below)
Column. CellStyleSet	(overrides all below)
Column. StyleSet	(overrides all below)
Group.StyleSet	(overrides all below)
Control.StyleSet	

The **RowSelectionStyle** property determines how selected rows appear. The style specified by **RowSelectionStyle** always overrides a StyleSet style for selected rows.

See Also

HeadStyleSet property, StyleSet property, StyleSet object , StyleSets collection

Add Method

Applies To

Bookmarks collection, Buttons collection, Columns collection, Groups collection, SelBookmarks collection, StyleSets collection

Description

Adds an object to a collection.

Syntax (*SSDBData: Bookmarks*)

Adds a bookmark to the **Bookmarks** collection of the SSDBData control.

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

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

Syntax (*SSDBCombo, SSDBDropDown, SSDBGrid: Columns/Groups*)

Adds a **Column** or **Group** object to the **Columns** or **Groups** collection, respectively, of the indicated control.

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

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

Remarks

For the **Columns** and **Groups** collections, which are 0-based, *index* cannot exceed the number of columns or groups, respectively, that currently exist in *object*. Additionally, you can insert *object* between two existing objects. 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
```

Referencing a **Column** or **Group** object that does not exist will add the object to the appropriate collection.

Syntax *(SSDBCombo, SSDBDropDown, SSDBGrid: SelBookmarks)*

Adds a bookmark to the **SelBookmarks** collection of the indicated control.

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
bookmark	A variant specifying the bookmark to add.

Remarks

The row specified by *bookmark* becomes visually highlighted when it is added to the collection. Depending on the value of *object*'s **SelectTypeRow** property, other rows may be removed from the collection when adding a row.

Syntax (*SSDBOptSet:Buttons*)

Adds a **Button** object to the **Buttons** collection of the indicated SSDBOptSet control.

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

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

Remarks

If a value for *ButtonsToAdd* is not specified, a single button will be added.

Syntax (*StyleSets*)

Adds a **StyleSet** object to the **StyleSets** collection of the indicated control.

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

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

Remarks

Referencing a **StyleSet** object that does not exist will add the referenced **StyleSet** object to the **StyleSets** collection.

See Also

Bookmarks collection, Buttons collection, Columns collection, Groups collection, StyleSets collection, Count property, Remove method, RemoveAll method

Example (SSDBData Bookmarks Collection)

The following code adds the bookmark of the current row in the recordset to the Bookmarks Collection.

```
SSDBData1.Bookmarks.Add Data1.Recordset.Bookmark, "The Row"
```

Example (Columns)

The following code adds 3 Columns to a DataGrid.

```
Dim i As Integer

For i = 0 To 2
    SSDBGrid1.Columns.Add i
Next i
```

Example (SelBookmarks)

The following code selects every row in the grid by adding their bookmarks to the SelBookmarks Collection.

```
Dim i As Integer

SSDBGrid1.MoveFirst
For i = 0 To SSDBGrid1.Rows - 1
    SSDBGrid1.SelBookmarks.Add SSDBGrid1.GetBookmark(i)
Next i
```

Example (SSDBOptSet)

This example adds five buttons to the buttons collection of an SSDBOptSet control:

```
SSDBOptSet1.Buttons.Add 5
```

Example (StyleSets)

The following code creates a new styleset called "Red" and sets it ForeColor property.

```
SSDBGrid1.StyleSets.Add "Red"
SSDBGrid1.StyleSets("Red").ForeColor = vbRed
```

AddItem Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Adds a row of data to an AddItem grid.

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
item	A string expression containing delimited text specifying the data to add. The text contained in the string is used to populate the row.
index	An integer expression specifying the number of the absolute row to which the string of data should be added. *index* cannot exceed the value of the **Rows** property.

Remarks

The **FieldSeparator** and **FieldDelimiter** properties are used to parse the text and add the data to the appropriate columns in the new row.

When no index is specified, a row containing the data is added to the end of the grid.

See Also

DataMode , FieldDelimiter , FieldSeparator , RemoveItem method

Example

In the following code, rs is a recordset containing an indefinite number of records. The recordset has three fields: LastName, FirstName, and Salary. The code populates the grid with these three fields, in order.

```
Dim AddItemString As String

SSDBGrid1.FieldSeparator = vbTab
rs.MoveFirst
Do Until rs.EOF
   AddItemString = rs!LastName
   AddItemString = AddItemString & vbTab
   AddItemString = AddItemString & rs!FirstName
   AddItemString = AddItemString & vbTab
   AddItemString = AddItemString & rs!Salary
   SSDBGrid1.AddItem AddItemString
   rs.MoveNext
Loop
```

AddItem Method (Column Object)

Applies To

Column Object

Description

Adds a string to the column's combo box.

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
item	A string expression specifying the data to add.
index	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.

See Also

Style, Columns collection

Example (Column Object)

The following example adds the state abbreviation NY to a combo box:

```
SSDBGrid1.Columns(3).AddItem "NY"
```

AddItemBookmark Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Returns a bookmark for a given absolute row number.

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
RowIndex	A numeric expression specifying the absolute row number. Valid range is from 0 to one less the value of *object*'s **Rows** property.

Remarks

This method essentially converts an absolute row number to a bookmark, which is generally more flexible.

The **AddItemBookMark** method only applies to AddItem mode.

RowIndex expects an absolute row number, not relative or visible row number, therefore the **Row** property cannot be used here.

The **AddItemRowIndex** method complements this method by providing the opposite functionality; it converts a bookmark into a row number.

See Also

AddItemRowIndex

Example

The following code shows how to move to a specific row of an additem grid.

```
Dim bm As Variant

bm = SSDBGrid1.AddItemBookmark(20)
SSDBGrid1.Bookmark = bm
```

AddItemRowIndex Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Returns the absolute row number for a given bookmark.

Syntax

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

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

Remarks

This method essentially converts bookmarks in AddItem mode to absolute row numbers, which range from 0 to one less the value of *object*'s **Rows** property.

The **AddItemRowIndex** method only applies to AddItem mode.

The **AddItemBookMark** method complements this method by providing the opposite functionality; it converts an absolute row number into a bookmark.

See Also

AddItemBookmark

Example

The following code reads out the current absolute row number of an additem grid and stores it in a variable.

```
Dim RowNumber As Long

RowNumber = SSDBGrid1.AddItemRowIndex(SSDBGrid1.Bookmark)
```

AfterClick Event

Applies To

SSDBCommand, SSDBData

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.

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
```

AfterColUpdate Event

Applies To

SSDBGrid

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**.

Remarks

Because this event is generated after the record has been updated, you should use the **BeforeColUpdate** event to validate or modify any data before it is written to the database.

See Also

AfterInsert event, AfterUpdate event, BeforeColUpdate event, BeforeDelete event, BeforeInsert event, BeforeUpdate event

AfterDelete Event

Applies To

SSDBGrid

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.

Remarks

By default, if an error occurs while attempting to delete the current record, a message box will be displayed to the user. Set *RtnDisperrMsg* to **False** (or 0) to prevent this dialog from being shown.

See Also

AfterColUpdate event, AfterInsert event, AfterUpdate event, BeforeColUpdate event, BeforeDelete event, BeforeInsert event, BeforeUpdate event

Example

This example shows how to use the **AfterDelete** event to execute custom code:

```
Private Sub SSDBGrid1_AfterDelete(RtnDispErrMsg As Integer)

   If RtnDispErrMsg = False Then
      MsgBox "Record deleted!"
   Else
      MsgBox "WARNING! Record could not be deleted!"
   End If
```

```
End Sub
```

AfterInsert Event

Applies To

SSDBGrid

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.

See Also

AfterColUpdate event, AfterDelete event, AfterUpdate event, BeforeColUpdate event, BeforeDelete event, BeforeInsert event, BeforeUpdate 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
```

AfterPosChanged Event

Applies To

SSDBGrid

Description

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

Syntax

Sub control_**AfterPosChanged** (**ByVal***WhatChanged* **As Integer, ByVal** *NewIndex* **As Integer**)

The event parameters are:

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

Settings

The settings for *WhatChanged* are:

Setting	Description
0	Column was moved individually.
1	Column was moved as part of a group movement.
2	Column was moved as a result of being individually swapped.
3	Column was moved as a result of its group being swapped.

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.

See Also

ColMove event, ColSwap event, GrpMove event, GrpSwap event

AfterUpdate Event

Applies To

SSDBGrid

Description

Occurs after the user updates the current row.

Syntax

Sub control_**AfterUpdate** (*RtnDispErrMsg* **As Integer**)

The event parameters are:

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

Remarks

By default, if an error occurs while attempting to update the current record, a message box will be displayed to the user. Set *RtnDisperrMsg* to **False** (or 0) to prevent this dialog from being shown.

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

See Also

AfterColUpdate event, AfterDelete event, AfterInsert event, BeforeColUpdate event, BeforeDelete event, BeforeInsert event, BeforeUpdate event

Example

The following code checks to see if Column 0 in the updated row of the grid is higher than 50. If it is, a command button is enabled. If not, the command button is disabled.

```
Private Sub SSDBGrid1_AfterUpdate(RtnDispErrMsg As Integer)
    If SSDBGrid1.Columns(0).Value > 50 Then
        Command1.Enabled = True
    Else
        Command1.Enabled = False
    End If
End Sub
```

Alignment Property

Applies To

SSDBData, SSDBOptSet

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*]

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

Settings (SSDBData)

Setting	Description
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)

Setting	Description
0	(Default) Left Justify

1	Right Justify

There are constants available for the settings of this property.

See Also

CaptionAlignment property

Alignment Property (Column Object)

Applies To

Column Object

Description

Determines how the text will be aligned within the column.

Syntax

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

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

Settings

Setting	Description
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.

AlignmentPicture Property

Applies To

StyleSet Object

Description

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

Syntax

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

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

Settings

Setting	Description
0	(Default) Left
1	Right
2	Left Of Text
3	Right Of Text
4	Fit To Caption
5	Tile

There are constants available for the settings of this property.

See Also

AlignmentText

AlignmentText Property

Applies To

StyleSet Object

Description

Determines how the text will be aligned.

Syntax

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

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

Settings

Setting	Description
0	Left Justify
1	Right Justify
2	Center Justify

There are constants available for the settings of this property.

See Also

AlignmentPicture

AllowAddNew Property

Applies To

SSDBGrid

Description

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

Syntax

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

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

Settings

Setting	Description
True	The user can add new records to the grid.
False	(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 (*).

See Also

AllowDelete, AllowUpdate

AllowColumnMoving Property

Applies To

SSDBGrid

Description

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

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

See Also

AllowColumnSizing , AllowColumnSwapping

AllowColumnShrinking Property

Applies To

SSDBGrid

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*]

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

Settings

Setting	Description
True	(Default) The user can shrink the group.
False	The user can not shrink the group.

Remarks

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

See Also

AllowGroupShrinking

AllowColumnSizing Property

Applies To

SSDBGrid

Description

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

Syntax

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

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

Settings

Setting	Description
True	(Default) Columns can be resized by the user.
False	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.

See Also

AllowColumnMoving , AllowColumnSwapping

AllowColumnSwapping Property

Applies To

SSDBGrid

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
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

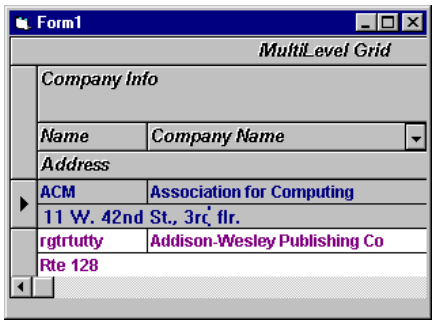
Remarks

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.

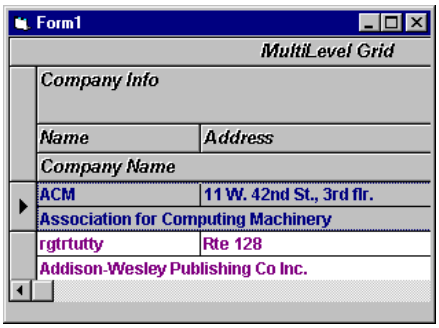
When columns are moved from one group to another, the column is removed from the source group and inserted in the destination group.

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:



Before Swap



After Swap

See Also

AllowColumnMoving , AllowColumnSizing

AllowDelete Property

Applies To

SSDBGrid

Description

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

Syntax

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

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

Settings

Setting	Description
True	Rows can be deleted by the user.
False	(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.

See Also

AllowAddNew, AllowUpdate

AllowDragDrop Property

Applies To

SSDBGrid

Description

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

Syntax

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

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

Settings

Setting	Description
True	(Default) Cell data can be moved by drag and drop.
False	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.

AllowGroupMoving Property

Applies To

SSDBGrid

Description

Determines if groups can be moved by the user.

Syntax

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

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

Settings

Setting	Description
True	(Default) Groups can be moved by the user.
False	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.

See Also

AllowGroupSizing, AllowGroupSwapping

AllowGroupShrinking Property

Applies To

SSDBGrid

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*]

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

Settings

Setting	Description
True	(Default) The user can shrink the group.
False	The user can not shrink the group.

Remarks

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

See Also

AllowColumnShrinking

AllowGroupSizing Property

Applies To

SSDBGrid

Description

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

Syntax

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

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

Settings

Setting	Description
True	(Default) Groups can be resized by the user.
False	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.

See Also

AllowGroupMoving, AllowGroupSwapping

AllowGroupSwapping Property

Applies To

SSDBGrid

Description

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

Syntax

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

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

Settings

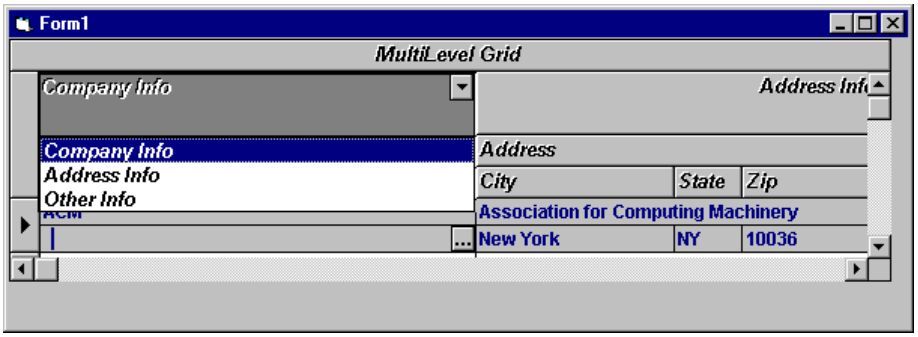
Setting	Description
True	(Default) Groups can be swapped by the user.
False	Groups can not be swapped by the user.

Remarks

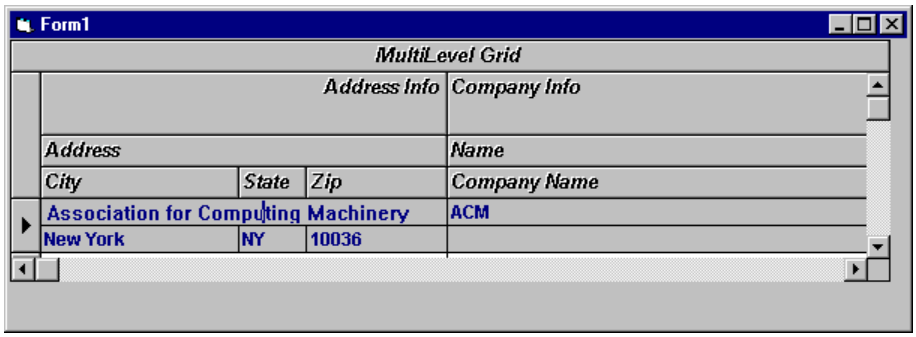
Swapping groups is accomplished by first clicking the header of one of the groups you wish to swap, then clicking on the dropdown list that appears, and finally selecting the name of the group you wish to swap with.

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



After Swap

See Also

AllowGroupMoving, AllowGroupSizing

AllowInput Property

Applies To

SSDBCombo

Description

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

Syntax

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

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

Settings

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

Remarks

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

See Also

AllowNull

AllowNull Property

Applies To

SSDBCombo

Description

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

Syntax

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

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

Settings

Setting	Description
True	(Default) SSDBCombo will allow a null value to be entered into the edit portion.
False	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.

AllowRowSizing Property

Applies To

SSDBGrid

Description

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

Syntax

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

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

Settings

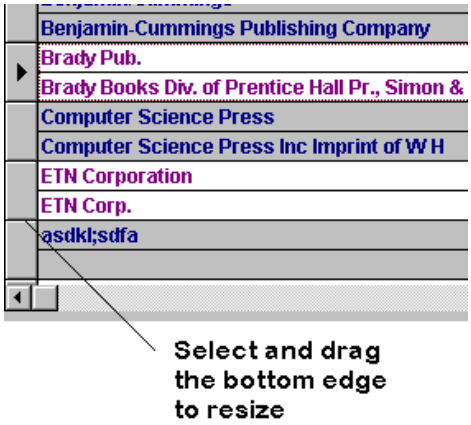
Setting	Description
True	(Default) Row height can be resized by the user.
False	Row height can not be resized by the user.

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.



AllowSizing Property

Applies To

Column object, Group object

Description

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

Syntax

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

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

Settings

Setting	Description
True	(Default) The object can be resized by the user.
False	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.

See Also

AllowColumnSizing

AllowUpdate Property

Applies To

SSDBGrid

Description

Determines if grid data can be modified by the user.

Syntax

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

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

Settings

Setting	Description
True	(Default) Data can be modified by the user.
False	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.

See Also

AllowAddNew, AllowDelete

AutoRestore Property

Applies To

SSDBCombo

Description

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

Syntax

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

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

Settings

Setting	Description
True	(Default) Text will be restored.
False	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.

AutoSize Property

Applies To

SSDBCommand

Description

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

Syntax

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

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

Settings

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

BackColor Property

Applies To

Column object, SSDBCombo, SSDBDropDown, SSDBData, SSDBGrid, SSDBOptSet

Description

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

Syntax

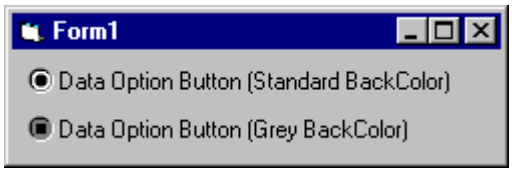
object . **BackColor**[= *color*]

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

Example

The following is an example of how the **BackColor** property affects the SSDBOptSet control:

```
SSDBOptSet1.Buttons(1).BackColor = &H00808080&
' Set to dark gray
```



BackColorEven Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

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

Syntax

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

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

See Also

BackColor, BackColorOdd

BackColorOdd Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

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

Syntax

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

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

See Also

BackColor, BackColorEven

BalloonHelp Property

Applies To

SSDBData, SSDBGrid

Description

Determines whether balloon help will be displayed.

Syntax

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

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

Settings

Setting	Description
True	(Default) Balloon Help will be displayed.
False	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.

BatchUpdate Property

Applies To

SSDBGrid

Description

Indicates to the OLE DB control whether Batch Optimistic record locking is being used by the data provider.

Syntax

object . **BatchUpdate**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether Batch Optimistic record locking is being used by the data provider, as described in Settings.

Settings

Setting	Description
True	The provider is using Batch Optimistic record locking (4 - adLockBatchOptimistic).
False	(Default) The provider is not using Batch Optimistic record locking..

Remarks

Because the control cannot determine what type of record locking the provider is utilizing, it is necessary to set this property to **True** when the provider is using Batch Optimistic record locking.

This property only applies to the OLE DB version of the control.

BeforeColUpdate Event

Applies To

SSDBGrid

Description

Occurs when a cell that has had its contents edited loses focus.

Syntax

Sub control_**BeforeColUpdate** ([*ColIndex* **As Integer**] [*OldValue* **As Variant**] [*Cancel* **As Integer**])

The event parameters are:

Parameter	Description
ColIndex	An integer expression that specifies the column to be updated.
OldValue	A variant that contains the value of the cell prior to any changes made by the user.
Cancel	An integer expression that specifies whether the operation occurs.

Remarks

This event is normally used to validate data entered in a cell. For row-level validation, use the **BeforeUpdate** event.

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

Note that the value returned by the *OldValue* parameter is always the value of the column when the row was entered. Even if the user makes multiple changes to the value of a column (with **BeforeColUpdate** being fired multiple times) the value returned by *OldValue* will remain the same until the row is exited.

By setting *Cancel* to **True**, focus from the control cannot be moved until the application determines that data can be moved back to the copy buffer.

You cannot programmatically move to another row in this event procedure (e.g., invoking the **MoveFirst** method will not position the grid at the first row).

See Also

AfterColUpdate, AfterDelete, AfterInsert, AfterUpdate, BeforeDelete, BeforeInsert, BeforeUpdate

Example (Cancel)

The following code will prevent the user from leaving a cell in the grid if the value in Column 0 is greater than 50.

```
If ColIndex = 0 Then
    If SSDBGrid1.Columns(0).Value > 50 Then
        Cancel = True
    End If
End If
```

Example (Calculation)

In the following code, Column 2 of the grid is updated with the total of Column 0 and 1.

```
If ColIndex = 0 Or ColIndex = 1 Then
    SSDBGrid1.Columns(2).Value = SSDBGrid1.Columns(0).Value _
        + SSDBGrid1.Columns(1).Value
End If
```

BeforeDelete Event

Applies To

SSDBGrid

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* to **True** (or -1) prevents the record from being removed.

See Also

AfterColUpdate, AfterDelete, AfterInsert, AfterUpdate, BeforeColUpdate, BeforeInsert, BeforeUpdate

Example

The following code replaces the default confirmation dialog with a custom dialog of the programmer's design.

```
Private Sub SSDBGrid1_BeforeDelete(Cancel As Integer, DispPromptMsg As
Integer)
    Dim Result As Long
    Dim PromptString As String

    'Cancel the default prompt
    DispPromptMsg = False

    'Display a confirmation msgbox
    PromptString = "You are about to delete "
    PromptString = PromptString & SSDBGrid1.SelBookmarks.Count
    PromptString = PromptString & _
        " rows. Are you sure you want to do this?"
    Result = MsgBox(PromptString, vbYesNo, "Delete?")
    Select Case Result
    Case vbYes
        'User chose to delete. Do nothing
    Case vbNo
        'User chose to Cancel
        Cancel = True
    End Select
End Sub
```

BeforeInsert Event

Applies To

SSDBGrid

Description

Occurs when the user makes a change to the Add Row, indicating that a new row of data is being created.

Syntax

Sub control_**BeforeInsert** ([*Cancel* **As Integer**])

The event parameters are:

Parameter	Description
Cancel	An integer expression that specifies whether the operation occurs.

Remarks

This event only applies when **AllowAddNew** is set to True.

The event is generated when the user makes a change to the Add Row, indicating that a new row of data is being created. You can use this event to prevent the addition of new rows to the database on an ad hoc basis, as determined by the state of the program at the time the user attempts to add the row.

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

The **BeforeUpdate** event is generated after the user exits the new row, but before the data is committed to the database. Only when the **AfterInsert** event takes place is the data moved from the buffer to the database.

Note Do not set the text of a cell in the event procedure for the **BeforeInsert** event. Doing so may cause a cascading event.

See Also

AfterColUpdate, AfterDelete, AfterInsert, AfterUpdate, BeforeColUpdate, BeforeDelete, BeforeUpdate

Example

The following code prevents the user from entering new rows in the grid, if the grid has 100 rows or more.

```
Private Sub SSDBGrid1_BeforeInsert(Cancel As Integer)
    Dim MsgString As String

    If SSDBGrid1.Rows > 99 Then
        Cancel = True
        MsgString = "You may not have more than 100 rows in this grid."
        MsgBox MsgString
    End If
End Sub
```

BeforeRowColChange Event

Applies To

SSDBGrid

Description

Occurs before the user moves to a different row or column.

Syntax

Sub control_**BeforeRowColChange** (*Cancel* **As Integer**])

The event parameters are:

Parameter	Description
Cancel	An integer expression specifying whether the row and/or column change will occur.

Remarks

This event is generated when the user attempts to leave the current cell, regardless of whether the cell was edited. Unless canceled, the **RowColChange** event is generated once the focus has moved to the new cell.

Setting *Cancel* to **True**, or -1, prevents the focus from changing, and halts execution of the **RowColChange** event.

See Also

BeforeColUpdate, RowColChange, RowLoaded

Example

The following code checks to see if the current cell in the grid is blank before the user leaves that cell. If it is blank, the user is prevented from moving off the cell.

```
Private Sub SSDBGrid1_BeforeRowColChange(Cancel As Integer)
    If SSDBGrid1.Columns(SSDBGrid1.Col).Text = "" Then
        Cancel = True
    End If
End Sub
```

BeforeUpdate Event

Applies To

SSDBGrid

Description

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

Syntax

Sub control_**BeforeUpdate** ([*Cancel* **As Integer**])

The event parameters are:

Parameter	Description
Cancel	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.

You can use this event to validate the data the user has entered into the Add row before they leave the row and commit the data.

Setting *Cancel* to **True** causes the update to not be written to the database, and prevents execution of the **AfterUpdate** event and the **AfterInsert** event.

This event will fire whenever the user moves off of a row that has been modified, even if the control is not bound to a database.

See Also

AfterColUpdate, AfterDelete, AfterInsert, AfterUpdate, BeforeColUpdate, BeforeDelete, BeforeInsert

BevelColorFace, BevelColorFrame, BevelColorHighlight, BevelColorShadow Properties

Applies To

SSDBCombo, SSDBCommand, SSDBData, SSDBDropDown, SSDBGrid, SSDBOptSet

Description

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

Syntax

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

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

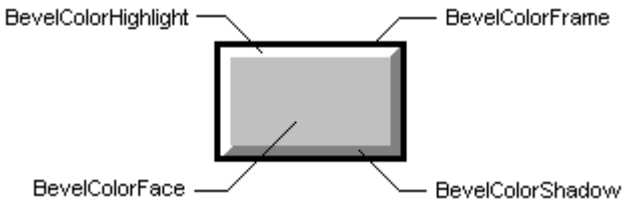
object . **BevelColorHighlight**[= *color*]

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
color	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.



BevelColorScheme Property

Applies To

SSDBCombo, SSDBCommand, SSDBData, SSDBDropDown, SSDBGrid, SSDBOptSet

Description

Determines the color scheme for the parts of the control.

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

See Also

BevelColorFace, BevelColorFrame, BevelColorHighlight, BevelColorShadow

BevelInner Property

Applies To

SSDBData

Description

Determines the type of inside beveling for the control.

Syntax

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

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

Settings

Setting	Description
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.

See Also

BevelColorFace, BevelColorFrame, BevelColorHighlight, BevelColorScheme, BevelColorShadow

BevelOuter Property

Applies To

SSDBData

Description

Determines the type of outside beveling for the control.

Syntax

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

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

Settings

Setting	Description
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.

See Also

BevelInner, BevelColorHighlight, BevelColorScheme, BevelColorShadow

BevelType Property

Applies To

SSDBCombo, SSDBDropDown

Description

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

Syntax

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

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

Settings

Setting	Description
0	None
1	(Default) Inset
2	Raised

There are constants available for the settings of this property.

BevelWidth Property

Applies To

SSDBCombo, SSDBCommand, SSDBData

Description

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

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

See Also

BevelInner, BevelOuter

Bold Property

Applies To

Font object, Headfont object

Description

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

Syntax

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

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

Settings

Setting	Description
True	Turns on bold formatting.
False	(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.

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 Object

Applies To

Bookmarks collection

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 Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Sets or returns a bookmark that uniquely identifies the current record.

Syntax

object . **Bookmark** [= *variant*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
variant	A variant expression specifying the bookmark value.

Remarks

Use the **Bookmark** property to save a reference to the current row. The reference remains valid even after another row becomes current. This is very useful when attempting to return to a row that was once selected.

See Also

AddItemBookmark method, RowBookmark method

Example

The following example uses the bookmark property of the grid in order to store a row and come back to it later.

```
'Store the current bookmark of the grid
bm = SSDBGrid1.Bookmark
'Move to the First Row in the Grid
SSDBGrid1.MoveFirst
'Make changes to the current (first) row
SSDBGrid1.Columns(0).Text = "Hello"
SSDBGrid1.Columns(1).Text = "World!"
'Commmit the changes
SSDBGrid1.Update
'Go back to the original row
SSDBGrid1.Bookmark = bm
```

Bookmark Property (ssRowBuffer only)

Applies To

ssRowBuffer object

Description

Sets or returns the bookmark value of the ssRowBuffer object.

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
index	The row in the ssRowBuffer object to which the bookmark refers.
variant	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 *index*, which is always a value from 0 to 9.

See Also

ReadType property, ssRowBuffer object, UnboundReadData event

BookmarkDisplay Property

Applies To

SSDBData

Description

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

Syntax

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

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

Settings

Setting	Description
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.

See Also

BookmarksToKeep

Bookmarks Collection

Applies To

SSDBData

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.

See Also

Bookmark Object

BookmarksToKeep Property

Applies To

SSDBData

Description

Sets or returns the maximum number of bookmarks to keep.

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

See Also

BookmarkDisplay

BorderStyle Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Sets or returns the border style of the control.

Syntax

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

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

Settings

Setting	Description
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 **SSDBDropDown**, this property affects the dropdown portion only.

BorderWidth Property

Applies To

SSDBData

Description

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

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

See Also

BevelInner, BevelOuter, BevelColorFace

BtnClick Event

Applies To

SSDBGrid

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.

Button Object

Applies To

Buttons collection

Description

The button object represents a button in the DataOptionSet.

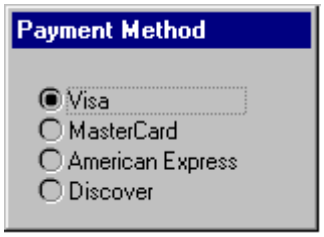
Properties

Caption	Picture	Value
ColOffSet	PictureMetaHeight	Visible
Enabled	PictureMetaWidth	
OptionValue	RowOffSet	

Remarks

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

An example of a DataOptionSet with four buttons:



ButtonEnabled Property

Applies To

SSDBOptSet

Description

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

Syntax

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

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

Settings

Setting	Description
True	(Default) The option button will be enabled.
False	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
```

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.

Place the following code in the **Load** event of the form:

```
SSDBOptSet1.ButtonEnabled = False
```

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

Applies To

SSDBOptSet

Description

Returns a button with the specified caption.

Syntax

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

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

See Also

ButtonFromPos

Example

The following example looks for a button with the caption of "Visa" and hides it:

```
SSDBOptSet1.ButtonFromCaption ("Visa").Visible = False
```

ButtonFromPos Method

Applies To

SSDBOptSet

Description

Returns the button which resides at a particular location.

Syntax

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

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

Settings

The settings for *scale* are:

Setting	Description
0	Twips (Default)
1	Pixels

2	Container Coordinates
3	HiMetric

There are constants available for the settings of this parameter.

See Also

ButtonFromCaption method, WhereIs 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
```

Buttons Collection

Applies To

SSDBOptSet

Description

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

Properties

Count	Item

Methods

Add	Remove	RemoveAll

See Also

Button Object

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)
```

ButtonsAlways Property

Applies To

Column object

Description

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

Syntax

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

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

Settings

Setting	Description
True	Buttons will be shown at all times for cells that have a button style property.
False	(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 4.

See Also

Style

ButtonSize Property

Applies To

SSDBData

Description

Sets or returns the size of each button.

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

ButtonVisible Property

Applies To

SSDBOptSet

Description

Determines whether the selected option button is visible or hidden.

Syntax

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

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

Settings

Setting	Description
True	(Default) Object is visible.
False	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.

See Also

IndexSelected, Visible

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:

Caption Property

Applies To

Button object, Column object, Group object, SSDBCommand, SSDBData, SSDBOptSet

Description

Determines the caption for the selected object/control.

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
text	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.

See Also

CaptionAlignment, Picture, PictureAlignment

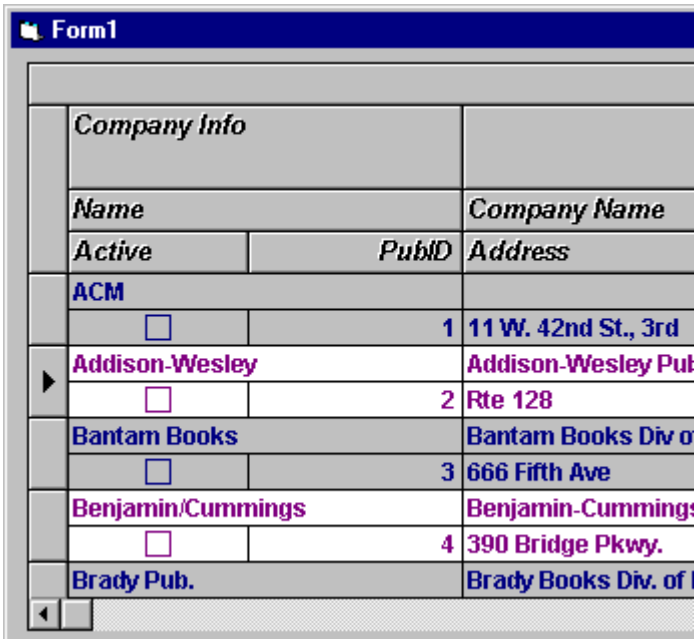
Example

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"
```



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"
```

CaptionAlignment Property

Applies To

Column object, Group object, SSDBData, SSDBGrid, SSDBOptSet

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*]

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

Settings (SSDBData, SSDBGrid, SSDBDropDown, Group Object)

Setting	Description
0	Left Justify

1	Right Justify
2	(Default) Center

There are constants available for the settings of this property.

Settings (Column Object)

Setting	Description
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)

Setting	Description
0	Right justify
1	(Default) Left justify

There are constants available for the settings of this property.

Settings (SSDBCommand)

Setting	Description
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.

See Also

PictureAlignment

Case Property

Applies To

Column object

Description

Sets or returns the case to use for column text.

Syntax

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

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

Settings

Setting	Description
0	(Default) Unchanged
1	lowercase
2	UPPERCASE

There are constants available for the settings of this property.

CellNavigation Property

Applies To

SSDBGrid

Description

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

Syntax

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

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

Settings

Setting	Description
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.

See Also

RowNavigation

CellStyleSet Method

Applies To

Column Object

Description

Sets the StyleSet for the specified cell.

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
StyleSet	A string expression specifying the name of the StyleSet to use.
RowNum	A variant expression specifying the number of a visible row (e.g., the first visble row is row 0) to which the StyleSet should be applied. Not specified when this method is invoked from within the **RowLoaded** event procedure.

Remarks

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

The **CellStyleSet** method behaves differently when utilized in the event procedure for the **RowLoaded** event. In the **RowLoaded** event procedure, *RowNum* is ignored (and should not be specified), since the StyleSet can only be set for the row that is currently being loaded.

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

Also note that the **CellStyleSet** method sets, but does not return the StyleSet for a particular cell, meaning that it cannot be used to determine which StyleSet is currently applied to a cell. This is because the grid does not actually store cell StyleSet information, since StyleSets are "virtually" applied to cells.

See Also

StyleSet, StyleSets collection

CellText Method

Applies To

Column Object

Description

Returns the underlying text for the specified cell.

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
Bookmark	A variant expression evaluating to a valid bookmark for a row in the control.

Remarks

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

The data comes directly from the underlying recordset in bound or unbound modes, and from the underlying memory buffer in AddItem mode. Therefore, cells that have had their value changed but have not been updated return their previous value instead.

The **CellValue** method returns the data in a format specified by the underlying data type. **CellText** always returns the data as a string.

In order to change the value of the underlying data, use the **Value** property.

See Also

CellValue method, Value property

Example

The following example takes the text from a cell and displays it in a message box:

```
DispText = SSDBGrid1.Columns(2).CellText(SSDBGrid1.Bookmark)
MsgBox "The value of Column 2 for the current row is :" + DispText)
```

CellValue Method

Applies To

Column object

Description

Returns the underlying data for the specified cell.

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
Bookmark	A variant expression evaluating to a bookmark for a row in the control.

Remarks

This method returns the value of the cell specified at the row represented by the bookmark in the format specifed by the underlying data.

The data comes directly from the underlying recordset in bound or unbound modes, and from the underlying memory buffer in AddItem mode. Therefore, cells that have had their value changed but have not been updated return their previous value instead.

The **CellText** method always returns the data as a string.

In order to change the value of the underlying data, use the **Value** property.

See Also

CellText method, Value property

Example

The following example uses the value of a cell to perform a calculation:

```
SalesTax = SSDBGrid1.Columns(5).CellValue(SSDBGrid1.Bookmark)
Total = (SubTotal * SalesTax)
```

Change Event

Applies To

SSDBCombo, SSDBGrid

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.

See Also

AfterUpdate, BeforeColUpdate, BeforeUpdate, BtnClick

CheckBox3D Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

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

Syntax

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

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

Settings

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

Click Event

Applies To

SSDBData

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

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.

ClipMode Property

Applies To

Column object, SSDBCombo

Description

Determines how mask characters will be copied to the clipboard.

Syntax

object.**ClipMode**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying whether to include mask literals in the data copied to the clipboard, as described in Settings.

Settings

Setting	Description
0	(Default) *ssIncludeMaskLiterals*. Literal characters will be copied to the clipboard. Clipboard text will match that displayed by the control.
1	*ssExcludeMaskLiterals*. Literal characters from the input mask will not be copied to the clipboard. Only the data will be copied.

There are constants available for the settings of this property.

Remarks

The **ClipMode** property controls how the data in a masked edit control is copied to the clipboard. If literals are included, the text will be copied to the clipboard just as it appears, for example "$20,000.00". If literals are excluded, only the data is copied, for example "20000"

This property has no effect if the **Mask** property is set to the empty string (""). This property has no effect on how the the **Text** property of the control, or whether literal characters are written to the database when the control is bound.

See Also

Mask, PromptChar, PromptInclude, ValidationError

ClippingOverride Property

Applies To

ssPrintInfo Object

Description

Sets or returns a value that determines whether to use extended clipping.

Syntax

object.**ClippingOverride**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying how to clip text, as described in Settings.

Settings	Description
0	(Default) *ssClippingOverrideAuto*. Automatic. The Data Widgets control will determine whether or not to use extended clipping.
1	*ssClippingOverrideYes*. Yes. The control will always use extended clipping.
2	*ssClippingOverrideNo*. No. The control will not use extended clipping.

There are constants available for the settings of this property.

Remarks

Sheridan Software has tested Data Widgets 3.1's printing capabilities with an array of hardware devices and device drivers. We have found inconsistencies in the way certain printer drivers implement the printing APIs that Data Widgets uses to create its reports. Data Widgets 3.1 can detect the presence of these drivers and compensate automatically.

Some printer drivers handle the clipping of text regions inconsistently. Without compensation, reports printed using these drivers may have text that overlaps multiple cells, or letters may extend below the grid lines dividing headers from rows, or rows from rows. Text wrapping may also be affected.

Note that, while rows can automatically resize themselves vertically to accommodate larger font sizes, column and group headers cannot. If the text in a column or group header is being clipped, you may simply need to resize the header so that all the text is showing. However, under normal circumstances the header text should never extend past the borders of the header, either vertically or horizontally.

The **ClippingOverride** property gives you the ability to determine how Data Widgets will handle the detection and correction of printer driver clipping inconsistencies. The default setting, *ssClippingOverrideAuto*, causes Data Widgets to automatically detect whether the current driver requires compensation, and apply it if it does. This setting should work in most cases, and should not be changed unless you absolutely have to.

The other settings of **ClippingOverride** give you manual control over whether clipping compensation is applied. Setting the property to *ssClippingOverrideYes* will apply text clipping compensation even if Data Widgets determines the driver does not require it. A setting of *ssClippingOverrideNo* will turn off compensation, even if Data Widgets determines that it is necessary.

Note An incorrect setting for the **ClippingOverride** property may produce unpredictable results when printing reports. You may see text overlapping the lines in the grid, text extending outside the cell that contains it, or you may see a gap between the edge of the printed text and the edge of the cell. If you experience problems such as these, first try setting **ClippingOverride** to its default setting. If you find these types of problems occurring when using the default value, try setting the property to one of the other values.

You should also note that problems of this nature may stem from problems completely external to Data Widgets and your application, such as insufficient printer memory. When attempting to troubleshoot printing problems, make sure your printer settings are correct and try using different driver settings (such as printing TrueType fonts as graphics or printing at a lower resolution) ***before*** changing the value of this property.

See Also

DriverOverride, PrinterDeviceName, PrinterDriverVer

CloseBookmarkDropDown Event

Applies To

SSDBData

Description

Occurs immediately before the dropdown bookmark list is closed.

Syntax

Sub control_**CloseBookmarkDropDown***(vBookmark* **As Variant**)

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

See Also

ShowBookmarkDropDown event

CloseFindDialog Event

Applies To

SSDBData

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
FindString	A Variant expression specifying the string to find.
Criteria	A Variant expression specifying the criteria to search for.
ColToSearch	A Variant expression specifying which column in the database to search.
Cancel	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.

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.

CloseUp Event

Applies To

SSDBCombo, SSDBDropDown

Description

Occurs when the dropdown portion of an SSDBCombo or SSDBDropDown control closes.

Syntax

Sub control_**CloseUp** ()

Remarks

This event is not generated for **Style** 3 (Combo Box) of a **Column** object, but for the SSDBCombo and SSDBDropDown controls. For the combo box style, the **ComboCloseUp** event is generated instead.

See Also

ComboDropDown event, ComboCloseUp event, DropDown event

Example

In the following example code, a DataGrid called SSDBGrid1 has two columns. Column 0 has a DataDropDown attached to it, called SSDBDropDown1. In the CloseUp event of the Dropdown, the code checks to see if the user selected "Locked" or "Unlocked" from the DropDown list and sets the Locked property of Column 1 of the grid accordingly.

```
Private Sub SSDBDropDown1_CloseUp()
    Select Case SSDBGrid1.Columns(0).Text
    Case "Locked"
        SSDBGrid1.Columns(1).Locked = True
    Case "UnLocked"
        SSDBGrid1.Columns(1).Locked = False
    End Select
End Sub
```

Col Property

Applies To

SSDBGrid

Description

Sets or returns the current column.

Syntax

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

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

Remarks

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

See Also

Grp, Row

ColChanged Property

Applies To

Column object

Description

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

Syntax

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

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

Settings

Setting	Description
True	(Default) The field has been modified.
False	The field has not been modified.

Remarks

This property is read-only.

Example

The following example demonstrates use of the **ColChanged** property

```
If SSDBGrid1.Columns(0).ColChanged Then
   MsgBox ("The name has changed!")
EndIf
```

ColContaining Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

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

Syntax

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

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

Remarks

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

Collate Property

Applies To

ssPrintInfo Object

Description

Determines the order in which multiple copies of pages will be printed.

Syntax

object.**Collate**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether copies will be collated, as described in Settings.

Settings

Setting	Description
True	Multiple copies will be collated when printed
False	(Default) Multiple copies will be printed in page order.

Remarks

Collating ensures that multiple copies of a printout emerge from the printer as complete individual documents. When collated, the first copy of a document will be printed in its entirety before the second copy of the document is begun. If you do not use collating, all the required copies of one page will be printed before the next page in the document is begun.

For example, if you print two copies of a three-page document, the collated document would be produced in this order: Page 1, Page 2, Page 3, Page 1, Page 2, Page 3. If you did not collate the document, the pages would be produced in this order: Page 1, Page 1, Page 2, Page 2, Page 3, Page 3.

This setting has no effect when printing a single copy of a document.

Example

The **PrintBegin** event fires just before the beginning of a print operation. You can use it to set various print options. Note that if a Print or Print Setup dialog was presented to the user, they have already had the opportunity to set these options, in which case the following code is not necessary unless you want to override one or more of the settings.

In this example, you explicitly set all of the print options.

If the printout is too wide to fit on one page, **PageBreakOnGroups** controls how to split the grid over multiple pages. Here you specify that the split should occur on group boundaries and that pages should be left-aligned. (Other options are to break on columns or to break on groups but center the data on the page).

```
Private Sub SSDBGrid1_PrintBegin(ByVal ssPrintInfo As ssPrintInfo)

   ssPrintInfo.Copies = 5           'Print 5 copies.
   ssPrintInfo.Collate = True       'The copies should be collated.
   ssPrintInfo.PageStart = 2        'Just print pages 2 through 5
   ssPrintInfo.PageEnd = 5
   ssPrintInfo.Portrait = True      'Print portrait style
                                    '(set to false for landscape)

   ssPrintInfo.PageBreakOnGroups = ssPrintLeftAlignGroups

   'Set margins.
   'If the U.S. (English) system is being used,
   'the setting will be in inches.
   'If the Metric system is being used,
   'the setting will be in millimeters.
   ssPrintInfo.MarginBottom = 1     'Set bottom margin to 1
   ssPrintInfo.MarginTop = 1        'Set top margin to 1
   ssPrintInfo.MarginLeft = 0.5     'Set left margin to 1/2
   ssPrintInfo.MarginRight = 0.5    'Set right margin to 1/2

End Sub
```

ColMove Event

Applies To

SSDBGrid

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
ColIndex	The column number being moved.
NewPos	An integer expression that specifies the visual position the column is being moved to.
Cancel	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.

See Also

AfterPosChanged event, ColSwap event, GrpMove event, GrpSwap event

ColOffSet Property

Applies To

Button object, SSDBOptSet

Description

Determines the horizontal offset used to draw the button.

Syntax

object . **ColOffSet**[= *number*]

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

Remarks

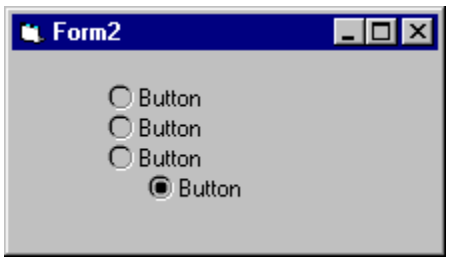
The value range for this property is -32767 to 32767 with a default value of 0.

See Also

RowOffSet

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
```

ColorMask Property

Applies To

SSDBData

Description

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

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
color	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.

See Also

ColorMaskEnabled

ColorMaskEnabled Property

Applies To

SSDBData

Description

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

Syntax

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

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

Settings

Setting	Description
True	The **ColorMask** property will affect the control.
False	(Default) The **ColorMask** 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).

See Also

ColorMask

ColPosition Method

Applies To

Group object, SSDBGrid

Description

Returns the index of the column relative to the collection.

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
ColPos	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.

See Also

Position, GrpPosition 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)
```

ColResize Event

Applies To

SSDBGrid

Description

Occurs before a column is resized.

Syntax

Sub control_**ColResize** ([*ColIndex* **As Integer**] [*Cancel* **As Integer**])

The event parameters are:

Parameter	Description
ColIndex	An integer expression specifying the column being resized.
Cancel	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.

See Also

GrpResize, RowResize, SplitterMove

Cols Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid, SSDBOptSet

Description

Sets or returns the number of columns in the control.

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

See Also

Col, Row, Rows

ColSwap Event

Applies To

SSDBGrid

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
ColIndex	The group number being moved.
NewPos	An integer expression that specifies the visual position the column is being swapped to.
Cancel	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.

See Also

AfterPosChanged event, ColMove event, GrpMove event, GrpSwap event

Column Object

Applies To

Columns collection

Description

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

Properties

Alignment	HasForeColor	NumberFormat
AllowSizing	HasHeadBackColor	Position
BackColor	HasHeadForeColor	PromptChar
ButtonsAlways	HeadBackColor	PromptInclude
Caption	HeadForeColor	Selected
CaptionAlignment	HeadStyleSet	Style
Case	ItemData *	StyleSet
ClipMode	Left	TagVariant
ColChanged	Level	Text
DataField	List	Top
DataType	ListCount	Value
DropDownhWnd	ListIndex	VertScrollBar
FieldLen	Locked	Visible
ForeColor	Mask	Width
Group	Name	
HasBackColor	Nullable	

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

Methods

AddItem	CellValue	RemoveItem

CellStyleSet	IsCellValid
CellText	RemoveAll

See Also

Columns collection

ColumnHeaders Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Determines whether column headers will be displayed.

Syntax

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

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

Settings

Setting	Description
True	(Default) Column headers will be displayed.
False	Column headers will not be displayed.

See Also

GroupHeaders

Columns Collection

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

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

Properties

Count	Item

Methods

Add	Remove	RemoveAll

See Also

Column object

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"
```

Columns Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Returns column object at specified index.

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
Index	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.

ColWidth Property

Applies To

DataOptSet

Description

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

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

ComboCloseUp Event

Applies To

SSDBGrid

Description

Occurs when a combo box is closed up.

Syntax

Sub control_**ComboCloseUp** ()

See Also

ComboDropDown event

ComboDropDown Event

Applies To

SSDBGrid

Description

Occurs when a combo box is dropped down.

Syntax

Sub control_**ComboDropDown** ()

See Also

ComboCloseUp event

ComboDroppedDown Property

Applies To

SSDBGrid

Description

Sets or returns the combo box's dropdown state.

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.

boolean	A Boolean expression specifying whether the combo box is dropped down, as described in Settings.

Settings

Setting	Description
True	Combo box is dropped down.
False	(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.

Copies Property

Applies To

ssPrintInfo Object

Description

Determines how many copies of the document will be printed.

Syntax

object.**Copies**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the number of copies to print.

Remarks

The **Copies** property determines how many copies of a data report will be printed. The default value of **Copies** is 1.

Count Property

Applies To

Bookmarks collection, Buttons collection, Columns collection, Groups collection, SelBookmarks collection, StyleSets collection

Description

Returns the total number of objects in the specified collection.

Syntax

object . **Count**

Part	Description
object	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.

See Also

Bookmarks collection, Buttons collection, Columns collection, Groups collection, StyleSets collection, Add method, Remove method, RemoveAll method

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)
```

DatabaseAction Property

Applies To

SSDBCommand

Description

Determines the action taken when the Data Command button is pressed.

Syntax

object . **DatabaseAction**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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 **PageValue** 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 **PageValue** property.
5	Go to the last record in the table.
6	Save a bookmark. Once saved, it is stored in the **SavedBookmark** 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 asumes 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.

DataField Property

Applies To

Column object, SSDBCombo, SSDBCommand, SSDBData, SSDBOptSet

Description

Determines a field to bind to in the current database.

Syntax

object . **DataField**[= *value*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
value	A string expression that evaluates to the name of the field in the object specified in the **DataSource** property.

Remarks

You must first specify a source in the **DataSource** property.

See Also

DataSource

DataFieldList Property

Applies To

SSDBCombo, SSDBDropDown

Description

Returns or sets a value that 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.

Syntax

object . **DataFieldList**[= *string*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
string	A string expression that evaluates to the name of one of the fields available in the data source bound to the control.

Remarks

When working in AddItem or unbound modes, columns are automatically assigned names, such as "Column 0" (for the first column) and "Column 1" (for the second). In these modes, it is the assigned names that should be used as the value of the **DataFieldList** property. Therefore, in order to assign the third column as the field used to return data, use "Column 2" (note the space).

The field specified by the **DataFieldList** property is used to determine the value of the SSDBCombo and SSDBDropDown's **Value** property. If the **DataFieldToDisplay** property is not set, the field is also used for the value of the control's **Text** property.

The **DataFieldToDisplay** property 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 **DataSourceList** property of the SSDBCombo control specifies the name of a valid data source, and the **DataFieldList** property specifies a valid field name in the data source.

Note The value specified for the **DataFieldList** property should be a unique (key) value. If you specify a data field for **DataFieldList** that contains duplicate entries, the incorrect entry may be stored in the database when the user selects a value from the dropdown.

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.**

See Also

DataFieldToDisplay, DataMode, DataSource, DataSourceList, Text, Value

DataFieldToDisplay Property

Applies To

SSDBCombo, SSDBDropDown

Description

Determines the field to display in the edit portion of the control.

Syntax

object . **DataFieldToDisplay**[= *value*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
value	A string expression that evaluates to the name of the field (in the object specified in the **DataSourceList** 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**. Otherwise, the field specified by the **DataFieldToDisplay** property is used, changing the value of the control's **Text** property.

Note The value specified for the **DataFieldToDisplay** property should be a unique (key) value. If you specify a data field for **DataFieldToDisplay** that contains duplicate entries, the incorrect entry may be stored in the database when the user selects a value from the dropdown.

When working in AddItem or unbound modes, columns are automatically assigned names, such as "Column 0" (for the first column) and "Column 1" (for the second). In these modes, it is the assigned names that should be used as the value of the **DataFieldList** property. Therefore, in order to assign the third column as the field used to return data, use "Column 2" (note the space).

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 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, 3, which can be used to 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.

See Also

DataField property, DataFieldList property, ReadType property, Text property

DataMember Property

Applies To

SSDBCombo, SSDBCommand, SSDBData, SSDBDropDown, SSDBGrid, SSDBOptSet

Description

Sets or returns a value that describes the data member for an OLE DB database connection.

Syntax

object . **DataMember**[= *value*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
value	A string expression that evaluates to the name of a member of the data environment specified by the **DataSource** property.

Remarks

This property is only available when using the OLE DB versions of the Data Widgets 3.1 controls!

This property determines the member of the data environment that will be bound to the control. The data environment to use is specified by the **DataSource** property. This property is not used when the value of the **DataSource** property refers to an ADO data control.

This property may be set at run time.

Once the **DataSource** and **DataMember** properties of the control have been properly set, the control is bound to a database through OLE DB.

For more information regarding ADO and OLE DB data providers, click here.

Note When binding the control to a database through OLE DB, you must distribute the SSR2C.DLL support file with your application.

See Also

DataMemberList, DataSource, DataSourceList

DataMemberList Property

Applies To

SSDBCombo

Description

Sets or returns a value that describes the data member for an OLE DB database connection through which the list portion of the current control is bound to a database.

Syntax

object . **DataMemberList**[= *value*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
value	A string expression that evaluates to the name of a member of the data environment specified by the **DataSourceList** property.

Remarks

This property is only available when using the OLE DB versions of the Data Widgets 3.1 controls!

This property determines the member of the data environment that will be used to populate the list portion of the control. The data environment to use is specified by the **DataSourceList** property. This property is not used when the value of the **DataSourceList** property refers to an ADO data control.

The **DataField** (edit portion), **DataFieldList** (list portion), and **DataSource** (edit portion) properties are utilized when connecting to a database via OLE DB as well as DAO/ODBC.

For more information regarding ADO and OLE DB data providers, click here.

Note When binding the control to a database through OLE DB, you must distribute the SSR2C.DLL support file with your application.

See Also

DataField, DataFieldList, DataMember, DataSource, DataSourceList

DataMode Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Sets/returns the mode used by the control for data access.

Syntax

object . **DataMode**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the data access mode used, as described in Settings.

Settings

Setting	Description
0	(Default) Bound mode
1	Unbound mode
2	AddItem mode

There are constants available for the settings of this property.

Remarks

Each data mode offers its own benefits. You should select the data mode that best fits your needs.

In bound mode, the control retrieves information from a database. Bound controls can be set up and maintained quickly and easily, but often are not as efficient as unbound controls. In this mode, properties, such as **DataSource**, **DataSourceList**, **DataMember**, and **DataMemberList**, are set to indicate the source of the data.

In unbound mode, the control receives data from another source, such as an array or unstructured source, which should be provided in the **UnboundReadData** event procedure. Unbound mode offers greater control over the data (such as virtual loading) and is more flexible and often more efficient. Generally, unbound mode provides a faster start time, but slower running speeds.

In AddItem mode, the control mimics an intrinsic control, and as such, is populated by invoking the **AddItem** method. In this mode, the entire set of data is provided to the control as the application starts, so generally AddItem mode is slower to start, but performs at better speeds while running.

It is possible, and often desired, to mix data modes, such as having a bound grid with an unbound column.

DataSource Property

Applies To

SSDBCombo, SSDBCommand, SSDBData, SSDBOptSet, SSDBDropDown, SSDBGrid

Description

Sets or returns a value that specifies the Data control (DAO data source), Data Environment connection or ADO Data control through which the current control is bound to a database.

Remarks

The purpose of this property varies depending on whether you are using the standard (DAO/ODBC) data binding version of the control or the OLE DB/ADO version of the control.

- *When using the DAO / ODBC version of the control:*

This property funcitons identically to the standard **DataSource** property supplied by the development environment. It is used to bind the control to an available recordset.

- *When using the OLE DB / ADO version of the control:*

This property specifies the OLE DB data environment to use with the control. Once the data environment has been established, you must specify which member of the data environment will be bound to the control by using the **DataMember** property. Once the **DataSource** and **DataMember** properties of the control have been properly set, the control will be bound to a database through OLE DB.

The **DataSource** property may be set at run time when using OLE DB binding, but not when using DAO binding.

For more information regarding ADO and OLE DB data providers, click here.

Note When binding the control to a database through OLE DB, you must distribute the SSR2C.DLL support file with your application.

See Also

DataMember, DataSourceList

DataSourceList Property

Applies To

SSDBCombo

Description

Sets a value that specifies the Data control (DAO data source), Data Environment connection or ADO Data control through which the list portion of the SSDBCombo control is bound to a database.

Remarks

The purpose of this property varies depending on whether you are using the standard (DAO/ODBC) data binding version of the control or the OLE DB/ADO version of the control.

- *When using the DAO / ODBC version of the control:*

Use this property 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. This property is not available at run time.

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.

- *When using the OLE DB / ADO version of the control:*

This property specifies the OLE DB data environment to use with the list portion of an SSDBCombo control. Once a data environment has been established, you must specify which member of the data environment will be bound to the list portion of the control by using the **DataMemberList** property. Once the **DataSourceList** and **DataMemberList** properties of the control have been properly set, the list portion of the control will be bound to

a database through OLE DB.

The **DataField** (edit portion), **DataFieldList** (list portion), and **DataSource** (edit portion) properties are utilized when connecting to a database via OLE DB as well as DAO/ODBC.

This property may be set at run time when using OLE DB binding, but not when using DAO binding.

For more information regarding ADO and OLE DB data providers, click here.

Note When binding the control to a database through OLE DB, you must distribute the SSR2C.DLL support file with your application.

See Also

DataField, DataFieldList, DataMemberList, DataMode, DataSource

DataType Property

Applies To

Column object

Description

Returns the column's underlying data type.

Syntax

object . **DataType**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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)

DefColWidth Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Sets or returns the default width used to initially display the column.

Syntax

object . **DefColWidth**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the width used to initially display the column.

DelayInitial Property

Applies To

SSDBCommand, SSDBData

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*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

See Also

DelaySubsequent

DelaySubsequent Property

Applies To

SSDBCommand, SSDBData

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*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

See Also

DelayInitial

Delete Event

Applies To

SSDBData

Description

Occurs when the user clicks the Delete button of the Enhanced Data Control.

Syntax

Sub control_**Delete** (*Cancel* **As Integer**, *DispPromptMsg* **As Integer**)

The event parameters are:

Parameter	Description
Cancel	An integer expression that specifes whether the operation occurs.
DispPromptMsg	An integer expression that indicates if a confirmation message box should be displayed to the user.

Remarks

Setting *Cancel* to **True** (or -1) cancels the deletion of the record.

Setting *DispPromptMsg* to **False** (or 0) prevents the confirmation message box from being displayed.

This event is not generated when a delete on the recordset is performed, only when the Delete button is clicked by the user.

DeleteSelected Method

Applies To

SSDBGrid

Description

Deletes all selected rows.

Syntax

object . **DeleteSelected**

Part	Description
object	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.

Invoking this method generates the **BeforeDelete** event.

See Also

BeforeDelete, SelBookmarks

DividerStyle Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Sets or returns the style of row divider used.

Syntax

object . **DividerStyle**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the divider style, as described in Settings.

Settings	Setting	Description
0	Black line	
1	(Default) Dark gray line	
2	Raised	
3	Inset	
4	ForeColor	

There are constants available for the settings of this property.

See Also

DividerType

DividerType Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Sets or returns the type of row divider used.

Syntax

object . **DividerType**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the divider type, as described in Settings.

Settings	Description
0	None
1	Vertical
2	Horizontal
3	(Default) Both

There are constants available for the settings of this property.

See Also

DividerStyle

DoClick Method

Applies To

SSDBCombo, SSDBCommand, SSDBDropDown, SSDBGrid

Description

Fires the **Click** event.

Syntax

object . **DoClick**

Part	Description
object	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.

DriverOverride Property

Applies To

ssPrintInfo Object

Description

Sets or returns a value that determines whether to override printer driver settings.

Syntax

object.**DriverOverride**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying how to override the printer driver, as described in Settings.

Settings	Description
0	(Default) *ssDriverOverrideAuto*. Automatic. The Data Widgets control will determine whether or not to override the print driver.
1	*ssDriverOverrideYes*. Yes. The control will always override the printer driver.
2	*ssDriverOverrideNo*. No. The control will not override the printer driver.

There are constants available for the settings of this property.

Remarks

Sheridan Software has tested Data Widgets 3.1's printing capabilities with an array of hardware devices and device drivers. We have found inconsistencies in the way certain printer drivers implement the printing APIs that Data Widgets uses to create its reports. Data Widgets 3.1 can detect the presence of these drivers and compensate automatically.

The **DriverOverride** property gives you the ability to determine how Data Widgets will handle the detection and correction of printer driver inconsistencies. The default setting, *ssDriverOverrideAuto*, causes Data Widgets to automatically detect whether the current driver requires compensation, and apply it if it does. This setting should work in most cases, and should not be changed unless you absolutely have to.

The other settings of **DriverOverride** give you manual control over whether compensation is applied. Setting the property to *ssDriverOverrideYes* will cause Data Widgets to always perform detection and apply correction if necessary. A setting of *ssDriverOverrideNo* will turn off compensation, even if Data Widgets determines that it is required.

Note An incorrect setting for the **DriverOverride** property may produce unpredictable results when printing reports. You may get multiple blank pages, pages containing garbage data, or find that multi-page reports contain progressively smaller printed areas on each consecutive page. If you experience problems such as these, first try setting **DriverOverride** to its default setting. If you find these types of problems occurring when using the default value, try setting the property to one of the other values.

See Also

ClippingOverride, PrinterDeviceName, PrinterDriverVer

DropDown Event

Applies To

SSDBCombo, SSDBDropDown

Description

Occurs when a dropdown drops down.

Syntax

Sub control_**DropDown** ()

See Also

CloseUp event

DropDownHwnd Property

Applies To

Column object

Description

Specifies the handle of the Data DropDown (Hwnd property) to be linked with the Data Grid.

Syntax

object . **DropDownHwnd**[= *hwnd*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
hwnd	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.

Example

The following code ties the second column of a Data Grid to a Data DropDown:

```
SSDBGrid1.Columns(1).DropDownHwnd = SSDBDropDown1.Hwnd
```

DroppedDown Property

Applies To

SSDBCombo, SSDBData, SSDBDropDown, SSDBGrid

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
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying the dropdown state, as described in Settings.

Settings (SSDBData)

Setting	Description
True	**ShowBookmarkDropdown** event is fired.
False	**CloseBookmarkDropdown** event is fired.

Settings (SSDBCombo/SSDBDropDown)

Setting	Description
True	When used in the **DropDown** event, cancels the dropdown.
False	No effect.

Settings (SSDBCombo/SSDBGrid)

Setting	Description
True	Causes the combo box to drop.
False	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.

See Also

CloseBookmarkDropdown event, ShowBookmarkDropdown event

Error Event

Applies To

SSDBGrid

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

This method is generated in bound mode only.

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."

See Also

AfterUpdate event, PrintError event, UpdateError event, How the Data Grid handles data validation and error checking

Export Method

Applies To

SSDBGrid

Description

Exports the contents of the grid to a file.

Syntax

object . **Export(***Type* **As Integer,** *Flags* **As Integer**, *ExportToFile* **As String**, [*HTMLTemplate* **As Variant**], [*OutputFileField* **As Variant**]**)**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
Type	An enumerated integer indicating the type of file to export, as described in Settings.
Flags	An integer expression specifying the options used for export, as described in Settings. Options are specified as a series of bit flags that may be combined using the OR operator. See Remarks for further details.
ExportToFile	A string expression specifying the filename of the export file.
HTMLTemplate	Optional. A variant expression specifying the filename of a template file that will be used to format the exported HTML file or files.
OutputFileField	Optional. A variant expression that specifies the name (or ordinal number) of the grid column

that contains a unique key that will be used to construct the output file name for each row file. Used only when *Type* is set to export row files.

Settings

The settings for *Type* are:

Setting	Description
0	*ssExportTypeText*. The data will be exported to an ASCII text file.
1	*ssExportTypeHTMLTable*. Data will be exported in table format to a single HTML file.
2	*ssExportTypeHTMLRowFiles*. Data will be exported as a series of HTML files, each with a table that contains a single row of data.

There are constants available for the settings of this parameter.

The settings for *Flags* are:

Setting	Description
1	*ssExportCurrentRow*. The row containing the active cell will be exported.
2	*ssExportSelectedRows*. All selected rows will be exported.
4	*ssExportNonSelectedRows*. All non-selected rows will be exported.
7	*ssExportAllRows*. (Combines bits 1, 2 and 4) - supplied as a convenience
16 (&H10)	*ssExportFieldNames*. Include the field names as a line in the output file.Used only when *Type* is set to '0 - ssExportTypeText.' Has no effect on HTML exports.
32 (&H20)	*ssExportColumnHeaders*. Include the column headers as a line in the output file.Used only when *Type* is set to '0 - ssExportTypeText.' Has no effect on HTML exports.
64 (&H40)	*ssExportInLayoutOrder*. Fields will be exported in the order that they are displayed by the grid. If this flag is not supplied, fields will be exported based on their ordinal positions in the **Columns** collection.
128 (&H80)	*ssExportHiddenColumns*. If specified, hidden columns will be exported. Otherwise, data from hidden columns will not appear in the output file.
256 (&H100)	*ssExportOverwriteExisting*. Replaces any existing output file with the same name as specified in *ExportToFile*.
512 (&H200)	*ssExportAppendToExisting*. If a file is found with the same name as specified in *ExportToFile*, exported data will be appended to the existing file. This option is primarily used when exporting text files. **Note** If neither *ssExportOverwriteExisting* or *ssExportAppendToExisting* are specified, an error will be generated if a file is found to exist with the same name as was specified in *ExportToFile*. If both are specified, any existing file will be overwritten.

There are constants available for the settings of this parameter.

Remarks

The **Export** method gives you the ability to take the data displayed in the grid and move it into an external file for further processing or display. Data can be exported to a plain ASCII text file, or it can be exported to an HTML file for use with a web browser. You specify the name of the file that will be created by passing it as the *ExportToFile* parameter of the method.

The export method can create three types of export files. For each type of file, you can determine whether to include some or all of the grid data in the output, and whether the generated file should replace an existing file of the same name, or be appended to it. You can also specify how field order will be determined during export. You specify these settings using the *Flags* parameter of the method. For any type of file, you can choose whether to

export all rows, selected rows, non-selected rows or the current row.

The values specified for the *Flags* parameter are a series of bit flags that selectively enable or disable combinations of options. You can specify multiple settings by combining the parameter values using logical OR. For example, specifying "ssExportFieldNames OR ssExportColumnHeaders" for the *Flags* parameter will include both field names and column headers in the exported text file.

The first type of file you can generate is a simple ASCII text file. Set the *Type* parameter of the method to *ssExportTypeText* (0) to generate a plain text file. You can specify export flags for the *Flags* parameter that specify whether to include group headers and/or field names as part of the output. The plain text export feature is useful for moving data into a third-party application, such as a spreadsheet. The **FieldDelimiter** and **FieldSeparator** properties are used when specifying this type of export.

The second type of export file is an HTML table file. When you specify *ssExportTypeHTMLTable* (1) for the *Type* parameter, the Data Grid will generate a complete HTML page based on the specified contents of the grid. Although you specify which rows will be exported using the *Flags* parameter, the remainder of the grid's formatting is determined by an HTML template file. You supply the filename of the template file as the *HTMLTemplate* parameter. The control reads the template file, extracts the HTML export codes embedded within the HTML and uses them to determine the formatting of the generated HTML.

You ***must*** specify a template file when you are exporting HTML. If you fail to specify a template file, no output will be generated. If you specify a template file when exporting plain text, the template will be ignored.

You must create HTML templates using an HTML or text editor. An extensive knowledge of HTML is not required, as the template format is fairly straightforward. Data Widgets includes several sample templates to get you started, and there is a tutorial that covers the creation and use of export templates in detail. For more information on creating HTML export templates, see HTML Template Codes.

The third type of export file is an HTML row file, created by specifying *ssExportTypeHTMLRowFiles* (2) for the *Type* parameter. This is similar to the HTML table file export, except that it generates multiple HTML files - one for each exported row of grid data. This is primarily useful in creating a master/detail web site, where clicking on a cell in the main table brings up a secondary page with related information about the data in that cell.

The names of the generated files are based on field in the data source; for example, if the records in the grid have an "ID" field with values of 12, 13, 14, etc., the **Export** method can automatically generate multiple files with names such as `ID_012.HTM`, `ID_013.HTM`, `ID_014.HTM`, etc. You implement this feature by specifying the field that will supply the numbers in the *OuptputFileField* parameter.

Note The method used to construct the output filenames varies depending on whether you are operating on a 16-bit or 32-bit platform. In a 16-bit environment, the name of the output file is taken from the first 4 letters of the file name specified as the *ExportToFile* parameter. This allows for up to 4 digits to be appended to the name of the generated file while still observing the 16-bit (8.3) filename length limit. In a 32-bit environment, support for long filenames eliminates the need to truncate the supplied output filename; the digits from the field specified by *OutputFileField* are simply appended to the filename specified in *ExportToFile*.

Once you invoke the **Export** method, the export begins. You can further monitor and control the export process using the **RowExport** event, which is fired once for each row of grid data exported into a file.

See Also

FieldDelimiter, FieldSeparator, HTML Template Codes, RowExport

Example

The following are two common uses of the Export method. The first exports grid data to a formatted HTML file, the second to a plain text file with column headers.

```
Private Sub cmdExport_Click()

   'Export all rows of the grid (exactly as it appears on screen)
   'to HTML using template.htm as the template file.
   SSDBGrid1.Export ssExportTypeHTMLTable, ssExportAllRows, _
   App.Path & "\Generated File.htm", App.Path & "\template.htm"

   'Export selected rows of the grid to a text file,
```

```
    'with the first line containing the field names.
    SSDBGrid1.Export ssExportTypeText, ssExportFieldNames & _
    ssExportSelectedRows, App.Path + "\Generated File.txt"

End Sub
```

FieldDelimiter Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Sets or returns the field delimiter used for an AddItem grid or exporting to a text file.

Syntax

object . **FieldDelimiter**[=*string*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
String	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".

The character specified by this property will be used as a field delimiter when the **Export** method is invoked to export the contents of a Data Grid to a text file.

See Also

FieldSeparator

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.

FieldLen Property

Applies To

Column object

Description

Sets or returns the maximum column field length for editing.

Syntax

object . **FieldLen**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

FieldSeparator Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Sets or returns the field separator used for an AddItem grid or exporting to a text file.

Syntax

object . **FieldSeparator**[= *text*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
text	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 separators is supplied, but you can enter your own. The default field separator is "tab".

The character specified by this property will be used as a field delimiter when the **Export** method is invoked to export the contents of a Data Grid to a text file.

See Also

FieldDelimiter

FieldValue Property

Applies To

SSDBData, SSDBOptSet

Description

Returns the field value for the active record.

Syntax

object . **FieldValue**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the field value.

Remarks

FieldValue contains the value of the field specified in **DataField** for the active record.

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)
```

Find Method

Applies To

SSDBData

Description

Finds a specified string in the database.

Syntax

object . **Find(***FindString* **as Variant**, *Criteria* **As Variant**, *Direction* **As Variant**, *ColToSearch* **As Variant)**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
FindString	Specifies the string to search for..
Criteria	Determines the criteria to use for the search, as described in Settings.
Direction	Determines the direction to search the database, as described in Settings.
ColToSearch	The index of the database column to search.

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.

See Also

CloseFindDialog event

FindBufferSize Property

Applies To

SSDBData

Description

Sets or returns the number of records read into the buffer for find operations.

Syntax

object . **FindBufferSize**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

See Also

FindDialog, ShowFindButtons

FindDialog Property

Applies To

SSDBData

Description

Determines whether the Find dialog is displayed.

Syntax

object . **FindDialog**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether the Find dialog is displayed, as described in Settings.

Settings	Setting	Description
True		The Find dialog is displayed.
False		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.

See Also

FindBufferSize, ShowFindButtons

FindFieldExclude Property

Applies To

SSDBData

Description

Determines which fields will not be displayed in the Find dialog box.

Syntax

object . **FindFieldExclude** = *fieldname* [;*fieldname*][;*fieldname*][...]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
fieldname	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.

See Also

FindFieldInclude

FindFieldInclude Property

Applies To

SSDBData

Description

Determines which fields will be displayed in the Find dialog box.

Syntax

object . **FindFieldInclude** = *fieldname* [;*fieldname*][;*fieldname*][...]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
fieldname	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.

See Also

FindFieldExclude

FindResult Event

Applies To

SSDBData

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 overriden by setting *RtnDispErrMsg* to **False**.

FirstRow Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Sets or returns the bookmark for the first visible row.

Syntax

object . **FirstRow**[= *variant*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
variant	A variant expression containing the bookmark of the first visible row. Note that this must be a bookmark, not a row number.

Remarks

This property is not available at design time. Setting this property makes the row referred to by the specified bookmark the first visible row in the grid.

Note that only bookmarks may be used in conjunction with this property; row numbers are not permitted.

Font Object

Applies To

SSDBCombo, SSDBCommand, SSDBData, SSDBDropDown, SSDBGrid, SSDBOptSet

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.

See Also

Bold, Font3D, Italic, Name, Size, Strikethrough, Underline

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

Applies To

SSDBCombo, SSDBCommand, SSDBData, SSDBDropDown, SSDBGrid, SSDBOptSet

Description

Determines the 3D style of the caption text for the control.

Syntax

object . **Font3D**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

See Also

Caption

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
```

ForeColorEven Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Determines the row's foreground (text) color for even-numbered rows.

Syntax

object . **ForeColorEven**[= *color*]

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

See Also

ForeColor, ForeColorOdd

ForeColorOdd Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Determines the row's foreground (text) color for odd-numbered rows.

Syntax

object . **ForeColorOdd**[= *color*]

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

See Also

ForeColor, ForeColorEven

GetBookmark Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Returns a bookmark of a row relative to the current row.

Syntax

object . **GetBookmark(** [*Row* **As Long**]**)**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
Row	A long value specifying the row offset of the bookmark to retrieve.

Remarks

This method is invoked to return a bookmark that refers to a row that is relatively positioned to the current row.

Positive values for *Row* refer to rows below the current row and negative values refer to rows above the current row.

If you are unfamiliar with the concepts used when working with bookmarks, you might want to peruse the Bookmarks Tutorial.

Examples

GetBookmark Method, GetBookmark Method (Bookmarks Tutorial)

Example

The following code displays the contents of the first column in the grid (column 0) to the immediate window.

```
Dim i As Integer
Dim bm As Variant

'Since GetBookmark requires a relative row number, a MoveFirst is
'necessary before the loop.
SSDBGrid1.MoveFirst
For i = 0 To SSDBGrid1.Rows - 1
    bm = SSDBGrid1.GetBookmark(i)
    Debug.Print SSDBGrid1.Columns(0).CellText(bm)
Next I
```

Group Object

Applies To

Groups collection

Description

The group object represents a group within a grid.

Properties

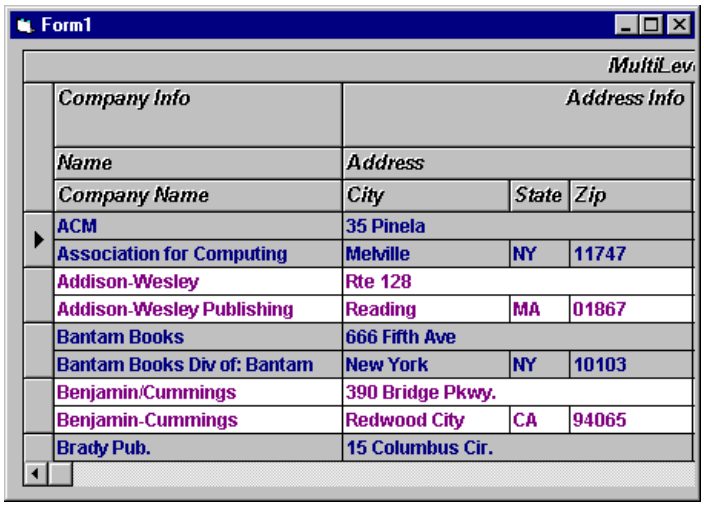
AllowSizing	HeadForeColor	Top
Caption	HeadStyleSet	Visible
CaptionAlignment	Left	Width
ColPosition	Position	
HasHeadBackColor	Selected	
HasHeadForeColor	StyleSet	
HeadBackColor	TagVariant	

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.



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 Property

Applies To

SSDBGrid

Description

Returns the current group.

Syntax

object . **Group**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the current group.

See Also

Col, Grp, Row

GroupHeaders Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Determines whether group headers will be displayed.

Syntax

object . **GroupHeaders**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether group headers will be displayed, as described in Settings.

Settings

Setting	Description
True	(Default) Group headers will be displayed.
False	Group headers will not be displayed

See Also

ColumnHeaders

GroupHeadLines Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Sets or returns the number of lines to display for group headers.

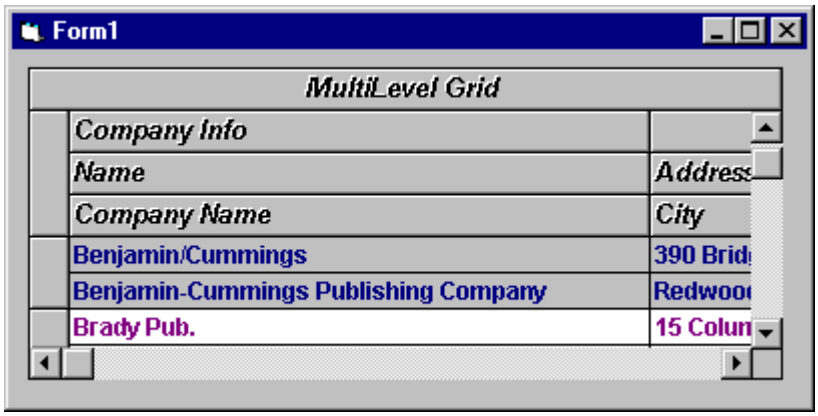
Syntax

object . **GroupHeadLines**[= *number*]

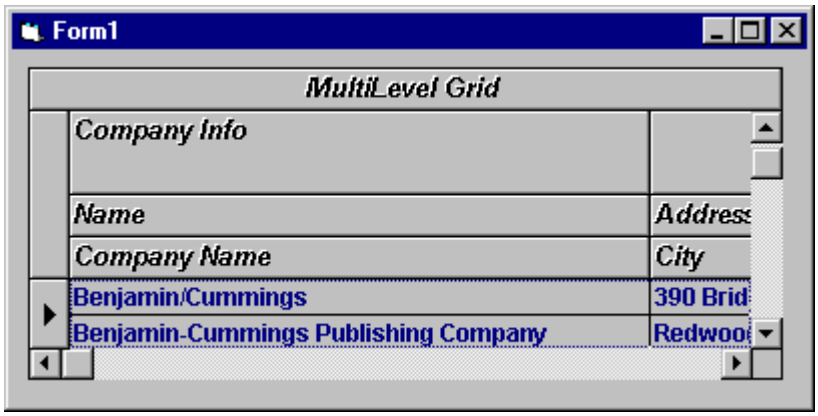
Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the number of lines to display for group headers.

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.

See Also

HeadLines

Groups Collection

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

The groups collection represents a set of data grid group objects.

Properties

Count Item

Methods

Add Remove RemoveAll

See Also

Group object

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"
```

Groups Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Returns group object at specified index.

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
Index	A variant specifying the group number.

Remarks

When no index is specified the group object is returned.

Grp Property

Applies To

Column object

Description

Sets or returns the current group.

Syntax

object . **Grp**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

See Also

Col, Group, Row

GrpContaining Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Returns the group number located at the specified x-coordinate.

Syntax

object . **GrpContaining(** *X* **As Single)**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
X	A floating point value specifying the X-coordinate

GrpHeadClick Event

Applies To

SSDBGrid

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.

See Also

GrpResize event, HeadClick 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
```

GrpMove Event

Applies To

SSDBGrid

Description

Occurs before a group is moved.

Syntax

Sub control_**GrpMove** (*GrpIndex* **As Integer,** *NewPos* **As Integer**, *Cancel* **As Integer**)

The event parameters are:

Parameter	Description
GrpIndex	The group number being moved.
NewPos	An integer expression that specifies the visual position the group is being moved to.
Cancel	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.

See Also

AfterPosChanged event, ColMove event, ColSwap event, GrpSwap event

GrpPosition Method

Applies To

SSDBGrid

Description

Returns the index of the group relative to the collection.

Syntax

object . **GrpPosition(***grppos* **As Integer)**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
grppos	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.

See Also

ColPosition method, Position property

Example

The following code returns the index for the group that appears third on the grid:

```
X=SSDBGrid1.GrpPosition (2)
```

GrpResize Event

Applies To

SSDBGrid

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.

See Also

GrpHeadClick, HeadClick, ResizeWidth

GrpSwap Event

Applies To

SSDBGrid

Description

Occurs before a group is swapped.

Syntax

Sub control_**GrpSwap** (*GrpIndex* **As Integer,** *NewPos* **As Integer**, *Cancel* **As Integer**)

The event parameters are:

Parameter	Description
GrpIndex	The group number being moved.
NewPos	An integer expression that specifies the visual position the group is being swapped to.

Cancel	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.

See Also

AfterPosChanged event, ColMove event, ColSwap event, GrpMove event

HasBackColor Property

Applies To

Column object

Description

Returns or sets whether the column has a background color.

Syntax

object . **HasBackColor**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether the column has its own background color, as described in Settings.

Settings

Setting	Description
True	Column has its own background color.
False	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.

See Also

HasForeColor

HasForeColor Property

Applies To

Column object

Description

Returns or sets whether the column has its own foreground color.

Syntax

object . **HasForeColor**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether the column has its own foreground color, as described in Settings.

Settings

Setting	Description
True	Column has its own foreground color
False	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 colulmn 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.

See Also

HasBackColor

HasHeadBackColor Property

Applies To

Column object, Group object

Description

Returns or sets whether the header has a background color specified.

Syntax

object . **HasHeadBackColor**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether the header has its own background color, as

described in Settings.

Settings

Setting	Description
True	Header has a background color specified.
False	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.

See Also

HasHeadForeColor, HeadBackColor, HeadForeColor

HasHeadForeColor Property

Applies To

Column object, Group object

Description

Returns or sets whether the header has a foreground color specified.

Syntax

object . **HasHeadForeColor**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether the header has a foreground color specified, as described in Settings.

Settings

Setting	Description
True	Header has a foreground color specified.
False	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.

See Also

HasHeadForeColor, HeadBackColor

HeadBackColor Property

Applies To

Column object, Group object

Description

Determines the header's background color.

Syntax

object . **HeadBackColor**[= *color*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
color	A long integer value or constant that determines the background color of the header.

See Also

HasHeadBackColor, HasHeadForeColor, HeadForeColor

HeadClick Event

Applies To

SSDBGrid

Description

Occurs when a column heading is clicked on.

Syntax

Sub control_**HeadClick** ([*ColIndex* **As Integer**])

The event parameters are:

Parameter	Description
ColIndex	The column number being clicked on.

See Also

GrpResize event, 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

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

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.

See Also

Bold, Font3D, Italic, Name, Size, Strikethrough, Underline

Example

The following code changes the **Bold** property setting of a **HeadFont** object:

```
SSDBGrid.HeadFont.Bold = True ' Sets caption/header text to bold
```

HeadFont3D Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Determines the 3D style of the caption and header text for the control.

Syntax

object . **HeadFont3D**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

HeadForeColor Property

Applies To

Column object, Group object

Description

Determines the column header foreground (text) color.

Syntax

object . **HeadForeColor**[= *color*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
color	A long integer value or constant that determines the foreground color of the column header.

See Also

HasHeadBackColor, HasHeadForeColor, HeadBackColor

HeadLines Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Sets or returns the number of lines to display for column headers.

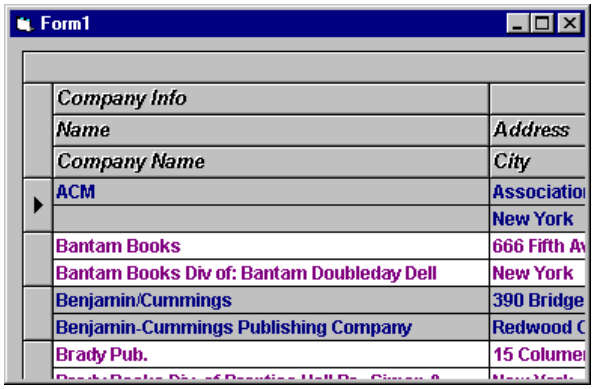
Syntax

object . **HeadLines**[= *number*]

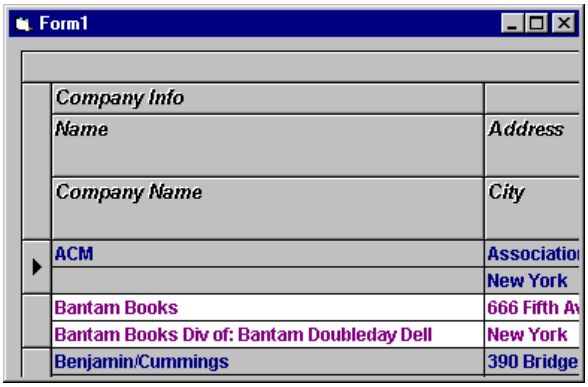
Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the number of lines to display for column headers.

Remarks

This example demonstrates the use of the **HeadLines** property:



This is an example of HeadLines = 1. The column headers (Name, Company Name, Address, City) each span one row.



This is an example of HeadLines = 2. The column headers (Name, Company Name, Address, City) each span two rows.

See Also

GroupHeadLines

HeadStyleSet Property

Applies To

Column object, Group object, SSDBCombo, SSDBDropDown, SSDBGrid

Description

Returns or sets the name of a **HeadStyleSet** in the **StyleSets** collection.

Syntax

object . **HeadStyleSet**[= *text*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
text	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

See Also

HeadStyleSet, StyleSet object, StyleSets collection

HeightGap Property

Applies To

SSDBOptSet

Description

Determines the amount of vertical distance between option buttons.

Syntax

object . **HeightGap**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

hWndEdit Property

Applies To

SSDBCombo, SSDBGrid

Description

Returns a handle to the edit portion of the grid.

Syntax

object . **HwndEdit**

Part	Description
object	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.

IndexSelected Property

Applies To

SSDBOptSet

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*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

InitColumnProps Event

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Occurs when the grid is initially loaded, allowing the setting of group and column properties.

Syntax

Sub control_**InitColumnProps** ()

Remarks

This event is provided so that you may initialize the control. It is recommended that you initialize the control (e.g., adding columns, groups, setting their properties, etc.) in this event procedure.

In bound mode, this event is generated when the control's DataControl is refreshed.

With the OLEDB version of the controls, this event is also generated when setting a datasource.

When adding unbound columns in this event procedure, the **UnboundReadData** event is not generated.

Example

The following example shows some common code that you might use in the **InitColumnProps** event of a DataGrid. The code adds three columns to the grid, makes the first column a CheckBox and attaches a DataDropDown to the second column.

```
Dim i As Integer

'Add three columns to the grid
For i = 0 To 2
   SSDBGrid1.Columns.Add i
Next i

'Make Column 0 a CheckBox Column
SSDBGrid1.Columns(0).Style = ssStyleCheckBox

'Attached a DataDropDown to Column 1 of the grid
SSDBGrid1.Columns(1).DropDownHwnd = SSDBDropDown1.hWnd
```

IsAddRow Method

Applies To

SSDBGrid

Description

Returns whether the current row is the add row.

Syntax

object . **IsAddRow**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether the current row is the add row.

Remarks

This method returns a Boolean value specifying whether the current row is the add row. This method is useful for determining if the current record is actually a new record being added.

Example

The following code uses the IsAddRow method to determine if a row is being added or edited when the BeforeUpdate event fires. It then adds a new row to the grid or updates an existing row in the grid accordingly.

```
Private Sub SSDBGrid1_BeforeUpdate(Cancel As Integer)
    Dim i As Integer

    If SSDBGrid1.IsAddRow Then
        rs.AddNew
    Else
        rs.FindFirst "PrimaryKey = " & SSDBGrid1.Columns(1).Value
        rs.Edit
    End If

    For i = 0 To 3
        rs.Fields(i).Value = SSDBGrid1.Columns(i).Value
    Next i
    rs.Update

End Sub
```

IsCellValid Method

Applies To

Column object

Description

Returns whether or not the text satisfies data type validation checking.

Syntax

object . **IsCellValid**

Part	Description
object	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.

See Also

IsTextValid method

IsItemInList Method

Applies To

SSDBCombo, SSDBGrid

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**

Part	Description
object	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.

See Also

IsTextValid method, ListAutoValidate property, ValidateList event

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
```

IsTextValid Method

Applies To

SSDBCombo

Description

Returns whether or not the text satisfies validation checking.

Syntax

object . **IsTextValid(***ErrCode* **As Variant)**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
ErrCode	An optional parameter that will return an error code describing the error if text is invalid.

Remarks

This method returns a Boolen value specifying whether or not the text could be validated.

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.IsTextValid then
         MsgBox "Text Is Not Valid!"
   EndIf
End Sub
```

See Also

IsCellValid, IsItemInList

Italic Property

Description

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

Syntax

object . **Italic**[= *boolean*]

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

Settings

Setting	Description
True	Turns on italic formatting.
False	(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.

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
```

LeftCol Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Sets or returns the left-most column in the display area of the grid.

Syntax

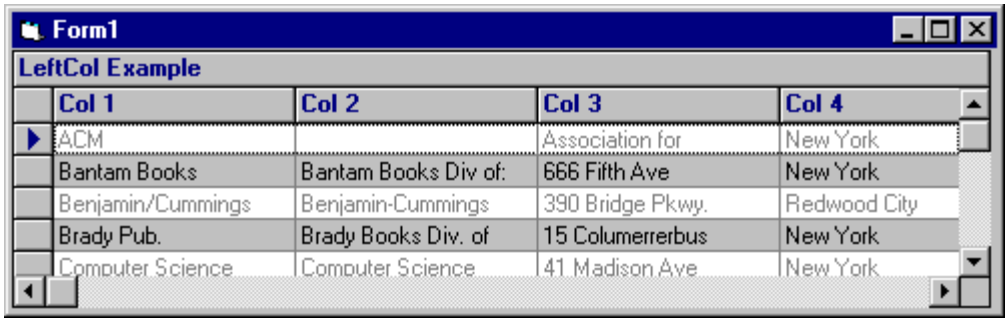
object . **LeftCol**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the left-most column.

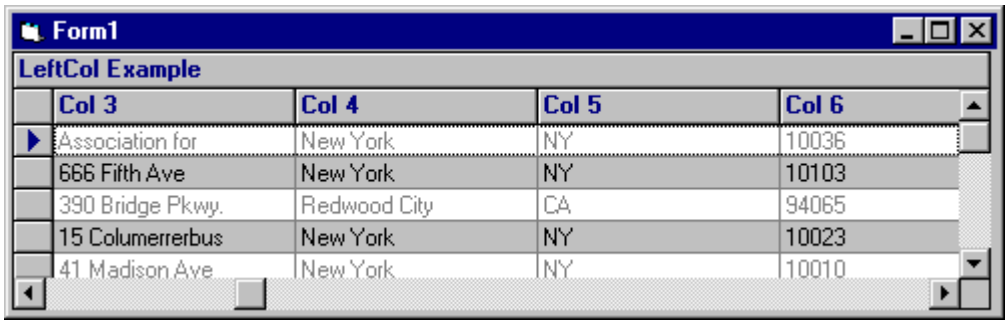
Note To ensure that your data displays accurately, **LeftCol** should only be used when you do not have groups defined in your grid. When you do have groups, use the **LeftGrp** property. This is due to the fact that the control bases its calculation on the column for this property; if you have groups, the resulting number will be invalid.

Remarks

The following example demonstrates LeftCol with different settings:



This is **LeftCol** set to 0



This is **LeftCol** set to 2

See Also

LeftGrp

LeftGrp Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Sets or returns the left-most group in the display area of the grid.

Syntax

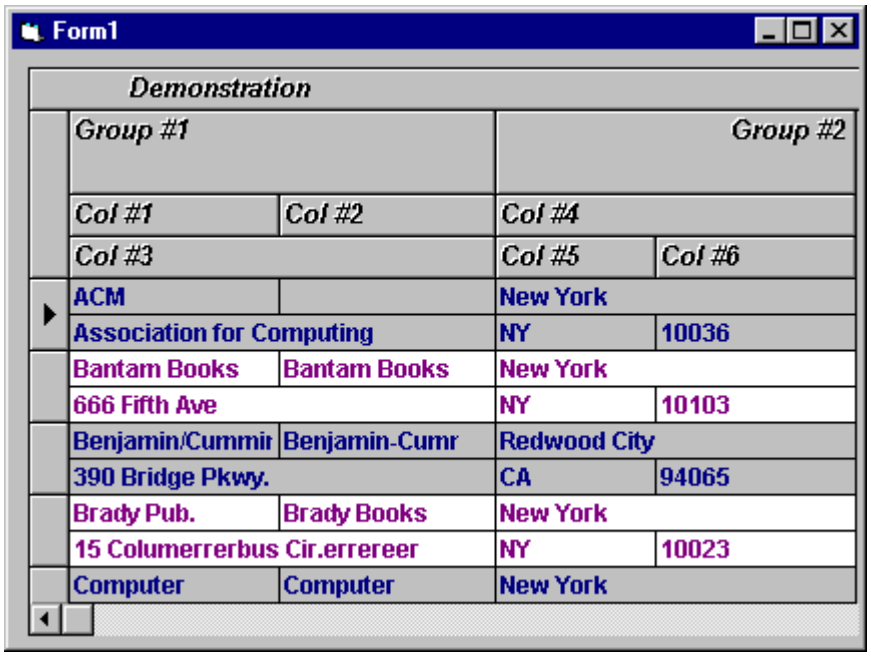
object . **LeftGrp**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the left-most group.

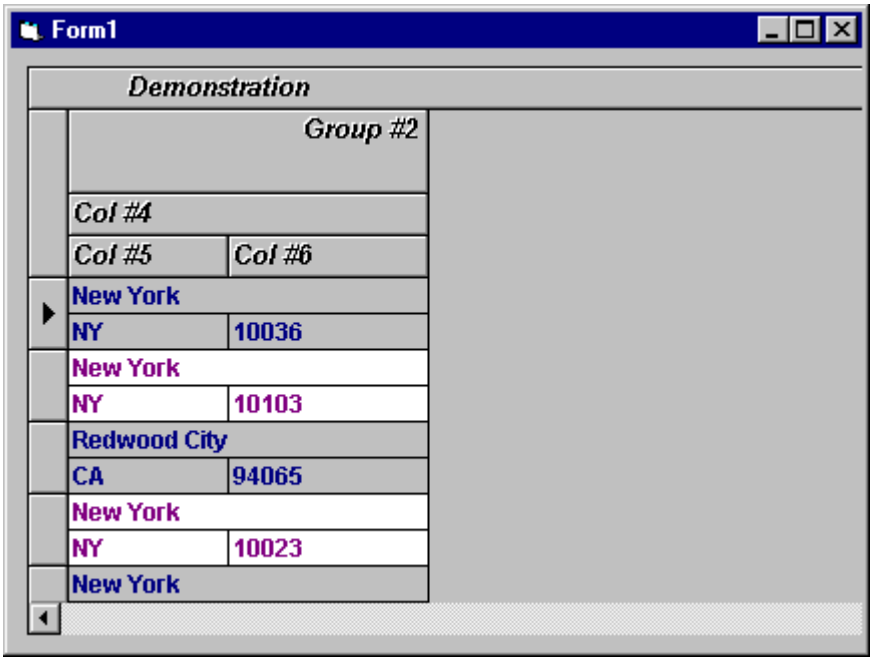
Remarks

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:



This is **LeftGrp** set to 0



This is **LeftGrp** set to 1

See Also

LeftCol

LevelCount Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Sets or returns the number of levels in a multi-level grid.

Syntax

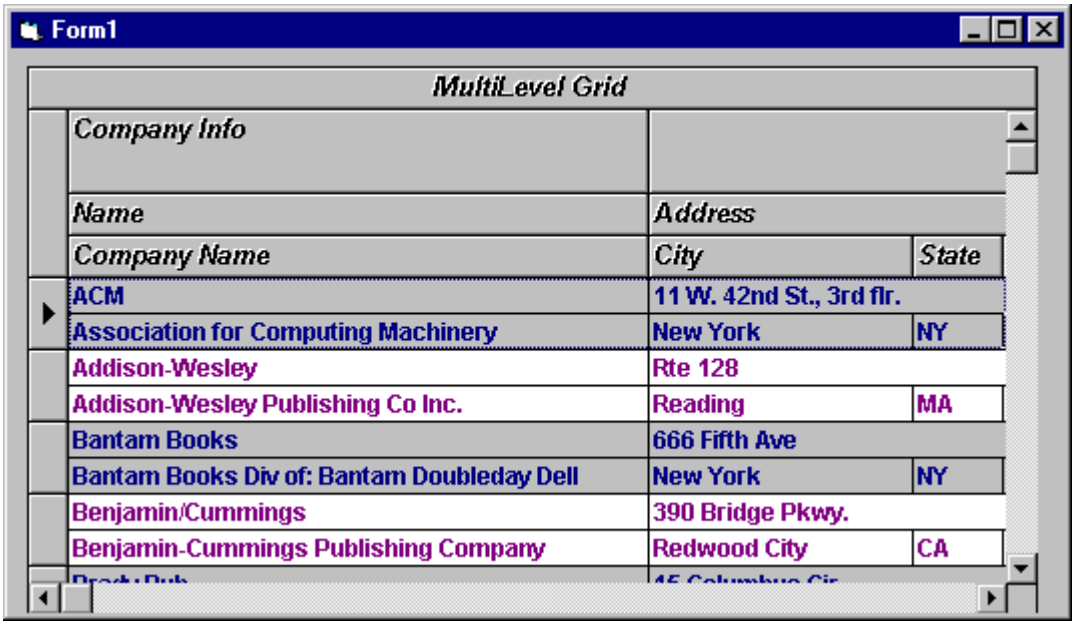
object . **LevelCount**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the number of levels in a multi-level grid.

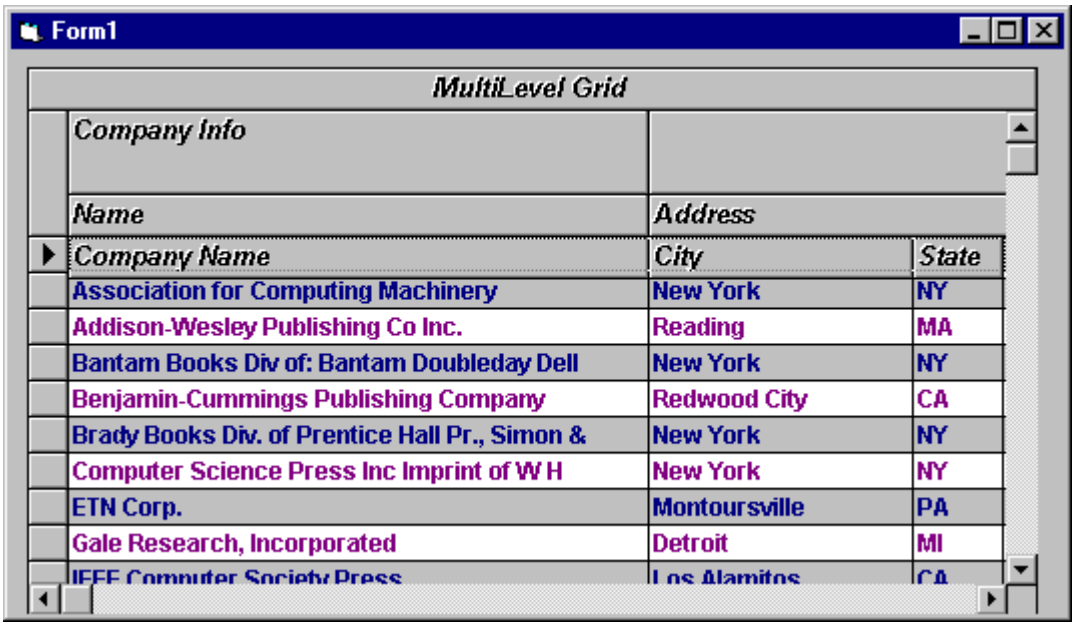
Remarks

Groups must be defined before using this property.

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.



In this example, **LevelCount** = 1. You will notice that each record is displaying on one row.

Level Property

Applies To

Column object

Description

Sets or returns the column's level within a multi-level grid.

Syntax

object . **Level**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the column's level in a multi-level grid.

List Property

Applies To

Column object

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**)

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
Index	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.

Example

The following example demonstrates the **List** property by adding two strings:

```
SSDBGrid1.Columns(2).List(3) = "Hello"
SSDBGrid1.Columns(2).List(4) = "World"
```

ListAutoPosition Property

Applies To

SSDBCombo, SSDBDropDown

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*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether the control should automatically position itself, as described in Settings.

Settings

Setting	Description
True	(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. The **PositionList** event will not be generated.
False	Disables this feature. The **PositionList** event will be generated so that positioning can be performed by code placed into the event procedure.

Remarks

The row the control selects is based upon the field specified by the **DataField** property, or, if set, the **DataFieldToDisplay** property.

Because record positioning is related to typing, much of its functionality is dependent upon the **AllowInput** property, for the SSDBCombo, and the **Locked** property of the linked Column when using the SSDBDropDown. When **AllowInput** is set to **True**, or **Locked** is set to **False**, record positioning only occurs when the dropdown portion of the control is displayed; otherwise, positioning can occur even when the dropdown portion is not displayed.

Additionally, if **AllowInput** is **True** or **Locked** is **False**, each letter typed adds to the match string used when positioning. For example, if three records exist, ABC Corp., Banana T-Shirts, and Charlie's Angle Fishing Co., typing the letters A, B, and C in succession would select the record for ABC Corp. However, when **AllowInput** is **False** or **Locked** is **True**, the matching is reset as each character is typed. Therefore, typing the same three letters as in the previous example would result in the record for Charlie's Angle Fishing Co. being selected because the last letter typed was C.

In cases where there is a large amount of rows in the list, or when using Unbound mode, you may want to consider setting this property to **False** for effeciency. You can then manually position the row in the list within the **PositionList** event procedure.

See Also

Performance Tuning

ListAutoValidate Property

Applies To

SSDBCombo, SSDBDropDown

Description

Determines whether the control will automatically check if the text entered in the edit portion is in the list.

Syntax

object . **ListAutoValidate**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether the text will automatically be validated, as described in Settings.

Settings

Setting	Description
True	(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.
False	Disables internal validation. The **ValidateList** event will be generated so that validation can be performed by code placed into the event procedure.

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.

The **ValidateList** event is only generated if this property is set to **False**.

For the SSDBCombo, the validation occurs (internally, if this property is set to **True**, in the **ValidateList** event, otherwise) if the position of the current record in a bound recordset changes or if the **IsItemInList** method is invoked.

For the SSDBDropDown, the validation occurs (internally, if this property is set to **True**, in the **ValidateList** event, otherwise) if the cell of the linked column in the SSDBGrid loses focus or either the **Update** or **IsItemInList** method is invoked.

If the entered value does not pass internal validation, the previous value is restored.

ListWidth Property

Applies To

SSDBCombo, SSDBDropDown

Description

Specifies the width of the control's entire list portion.

Syntax

object . **ListWidth**[= *single*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
single	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**.

See Also

ListWidthAutoSize

ListWidthAutoSize Property

Applies To

SSDBCombo

Description

Determines whether the control should automatically size the dropdown based on the number of columns in the list.

Syntax

object . **ListWidthAutoSize**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether the control should automatically size itself, as described in Settings.

Settings

Setting	Description
True	(Default) Causes the control to automatically size the dropdown based on the number of columns in the list.
False	Disables this feature.

Remarks

The width calculated will not exceed the total width of the screen.

LoadLayout Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Loads a previously saved layout, adding to or replacing the current grid layout.

Syntax

object . **LoadLayout**(*Filename* **As String**)

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
Filename	A string expression specifying the path to the layout file to be loaded.

Remarks

The **LoadLayout** method loads a saved layout file into the current grid. Saved layouts may include complete layout information or only StyleSet information. Loading a completely saved layout will replace the grid's current layout, including column and group arrangement, captions, data binding, colors, fonts, etc. (To find out which attributes of the control are overwritten when a new layout is loaded, see the **SaveLayout** method.)

The StyleSets in the saved layout will be added to the grid, and any existing StyleSets that have the same name as a saved StyleSet will be overwritten. Loading a layout which consists solely of saved StyleSets will add the StyleSets to the existing grid, overwriting any existing StyleSets that have the same name as those that were saved.

Grid layout files can have any name; the default extension for a grid layout file is .GRD. There is no difference between a layout file created at run-time and one created at design time using the Grid Editor. You may want to develop a naming convention for your layout files so you can differentiate between the ones that contain complete layout information and the ones that contain StyleSets only.

LoadLayout is primarily used to offer the user a way to save custom layouts they have created by sizing and moving groups and columns at run time. You can also use it to implement layout templates that give the user a variety of pre-formatted views for looking at their data. A third use is to store groups of StyleSets you have created and apply them across multiple controls and multiple projects.

Invoking **LoadLayout** through code is similar to clicking the "Load..." button in the Grid Editor and selecting a saved layout file. Saved layouts can be used at either design-time or run-time.

Note Fonts and pictures saved in a layout file by a 32-bit program cannot be read by a 16-bit program. If you create a .GRD file using the 32-bit version of DataWidgets, then load the file using the 16-bit version, font and picture information will not be applied.

It is possible to load a layout that was saved by a grid bound to a different data source. In this case, the columns saved from the original data source will be added to the current grid as unbound columns. You may then write code to rename the new columns or assign them to new data fields as needed.

Tip You may want to consider always saving the current layout using the **SaveLayout** method before invoking **LoadLayout**. This gives you the ability to offer the user an undo feature if the new layout does not meet their expectations.

See Also

SaveLayout

Example

This routine saves the current layout, loads a layout used for printing,then restores the original layout again. You may want to do this to use a different color scheme for printing than the one shown on the screen.

```
Private Sub PrintGridUsingPrintLayout()

   On Error GoTo layout_err_1

   'Save current grid layout.
   SSDBGrid1.SaveLayout App.Path + "\User Layout.grd", ssSaveLayoutAll
   'Load the print layout.
   SSDBGrid1.LoadLayout App.Path + "\Print Layout.grd"
   'Print the grid.
   SSDBGrid1.PrintData ssPrintAllRows, False, True
   'Restore the user's layout
   SSDBGrid1.LoadLayout App.Path + "\User Layout.grd"

Exit Sub

layout_err_1:
   MsgBox Err.Description

End Sub
```

Locked Property

Applies To

Column object

Description

Determines whether the column is locked from user-input.

Syntax

object . **Locked**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying the status of user-input in the column, as described in Settings.

Settings

Setting	Description
True	User input into the column is not allowed.
False	(Default) User input into the column is allowed.

Remarks

When tabbing between columns, columns that are locked will be skipped.

When this property is set to **True** for a column linked to an SSDBDropDown control (or the column's **Style** is set to ComboBox), the SSDBDropDown (or ComboBox) will allow selection only, and will not accept input. This is

similar to the behavior exhibited by the intrinsic ComboBox control when its **Style** property set to Dropdown List. Additionally, in terms of the list positioning that occurs when the dropdown portion is displayed, the match string is reset as each new character is typed. So, for example, if three records exist, ABC Corp., Banana T-Shirts, and Charlie's Angle Fishing Co., typing the letters A, B, and C in succession would select the record for Charlie's Angle Fishing Co. because the last letter typed was C.

Example

The following code demonstrates the Locked property by generating an error when a user attempts to click on a locked column.

```
Private Sub SSDBGrid1_Click()
   If SSDBGrid1.Columns(SSDBGrid1.Col).Locked = True Then
      MsgBox ("This field can not be edited!")
   End If
End Sub
```

MaintainBtnHeight Property

Applies To

SSDBOptSet

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*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether button height should be kept the same, as described in Settings.

Settings

Setting	Description
True	(Default) Buttons will automatically have the same height.
False	Buttons will not automatically have the same height.

MarginBottom Property

Applies To

ssPrintInfo Object

Description

Determines the bottom margin of a printed page of data.

Syntax

object.**MarginBottom**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	A single precision expression specifying the bottom margin of the printout.

Remarks

The **MarginBottom** property specifies the distance between the bottom of the printed text and the bottom of the page on a printout. This includes any text specified for the page footer.

The system of measurement used by the operating system (as specified by Regional Settings Properties in the Control Panel) will determine the units used by the **MarginBottom** property. If the U.S. (English) system is being used, the setting will be in inches. If the Metric system is being used, the setting will be in millimeters.

See Also

MarginLeft, MarginRight, MarginTop

Example

The **PrintBegin** event fires just before the beginning of a print operation. You can use it to set various print options. Note that if a Print or Print Setup dialog was presented to the user, they have already had the opportunity to set these options, in which case the following code is not necessary unless you want to override one or more of the settings.

In this example, you explicitly set all of the print options.

If the printout is too wide to fit on one page, **PageBreakOnGroups** controls how to split the grid over multiple pages. Here you specify that the split should occur on group boundaries and that pages should be left-aligned. (Other options are to break on columns or to break on groups but center the data on the page).

```
Private Sub SSDBGrid1_PrintBegin(ByVal ssPrintInfo As ssPrintInfo)

   ssPrintInfo.Copies = 5           'Print 5 copies.
   ssPrintInfo.Collate = True       'The copies should be collated.
   ssPrintInfo.PageStart = 2        'Just print pages 2 through 5
   ssPrintInfo.PageEnd = 5
   ssPrintInfo.Portrait = True      'Print portrait style
                                    ' (set to false for landscape)

   ssPrintInfo.PageBreakOnGroups = ssPrintLeftAlignGroups

   'Set margins.
   'If the U.S. (English) system is being used,
   'the setting will be in inches.
   'If the Metric system is being used,
   'the setting will be in millimeters.
   ssPrintInfo.MarginBottom = 1     'Set bottom margin to 1
   ssPrintInfo.MarginTop = 1        'Set top margin to 1
   ssPrintInfo.MarginLeft = 0.5     'Set left margin to 1/2
   ssPrintInfo.MarginRight = 0.5    'Set right margin to 1/2

End Sub
```

MarginLeft Property

Applies To

ssPrintInfo Object

Description

Determines the left margin of a printed page of data.

Syntax

object.**MarginLeft**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	A single precision expression specifying the left margin of the printout.

Remarks

The **MarginLeft** property specifies the distance between the left side of the printed text and the left edge of the page on a printout.

The system of measurement used by the operating system (as specified by Regional Settings Properties in the Control Panel) will determine the units used by the **MarginLeft** property. If the U.S. (English) system is being used, the setting will be in inches. If the Metric system is being used, the setting will be in millimeters.

See Also

MarginBottom, MarginRight, MarginTop

Example

The **PrintBegin** event fires just before the beginning of a print operation. You can use it to set various print options. Note that if a Print or Print Setup dialog was presented to the user, they have already had the opportunity to set these options, in which case the following code is not necessary unless you want to override one or more of the settings.

In this example, you explicitly set all of the print options.

If the printout is too wide to fit on one page, **PageBreakOnGroups** controls how to split the grid over multiple pages. Here you specify that the split should occur on group boundaries and that pages should be left-aligned. (Other options are to break on columns or to break on groups but center the data on the page).

```
Private Sub SSDBGrid1_PrintBegin(ByVal ssPrintInfo As ssPrintInfo)

   ssPrintInfo.Copies = 5           'Print 5 copies.
   ssPrintInfo.Collate = True       'The copies should be collated.
   ssPrintInfo.PageStart = 2        'Just print pages 2 through 5
   ssPrintInfo.PageEnd = 5
   ssPrintInfo.Portrait = True      'Print portrait style
                                    '(set to false for landscape)

   ssPrintInfo.PageBreakOnGroups = ssPrintLeftAlignGroups

   'Set margins.
   'If the U.S. (English) system is being used,
   'the setting will be in inches.
   'If the Metric system is being used,
   'the setting will be in millimeters.
   ssPrintInfo.MarginBottom = 1     'Set bottom margin to 1
   ssPrintInfo.MarginTop = 1        'Set top margin to 1
   ssPrintInfo.MarginLeft = 0.5     'Set left margin to 1/2
   ssPrintInfo.MarginRight = 0.5    'Set right margin to 1/2

End Sub
```

MarginRight Property

Applies To

ssPrintInfo Object

Description

Determines the right margin of a printed page of data.

Syntax

object.**MarginRight**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	A single precision expression specifying the right margin of the printout.

Remarks

The **MarginRight** property specifies the distance between the right edge of the printed text and the right edge of the page on a printout.

The system of measurement used by the operating system (as specified by Regional Settings Properties in the Control Panel) will determine the units used by the **MarginRight** property. If the U.S. (English) system is being used, the setting will be in inches. If the Metric system is being used, the setting will be in millimeters.

See Also

MarginBottom, MarginLeft, MarginTop

Example

The **PrintBegin** event fires just before the beginning of a print operation. You can use it to set various print options. Note that if a Print or Print Setup dialog was presented to the user, they have already had the opportunity to set these options, in which case the following code is not necessary unless you want to override one or more of the settings.

In this example, you explicitly set all of the print options.

If the printout is too wide to fit on one page, **PageBreakOnGroups** controls how to split the grid over multiple pages. Here you specify that the split should occur on group boundaries and that pages should be left-aligned. (Other options are to break on columns or to break on groups but center the data on the page).

```
Private Sub SSDBGrid1_PrintBegin(ByVal ssPrintInfo As ssPrintInfo)

   ssPrintInfo.Copies = 5           'Print 5 copies.
   ssPrintInfo.Collate = True       'The copies should be collated.
   ssPrintInfo.PageStart = 2        'Just print pages 2 through 5
   ssPrintInfo.PageEnd = 5
   ssPrintInfo.Portrait = True      'Print portrait style
                                    '(set to false for landscape)

   ssPrintInfo.PageBreakOnGroups = ssPrintLeftAlignGroups

   'Set margins.
   'If the U.S. (English) system is being used,
   'the setting will be in inches.
   'If the Metric system is being used,
```

```
    'the setting will be in millimeters.
    ssPrintInfo.MarginBottom = 1      'Set bottom margin to 1
    ssPrintInfo.MarginTop = 1         'Set top margin to 1
    ssPrintInfo.MarginLeft = 0.5      'Set left margin to 1/2
    ssPrintInfo.MarginRight = 0.5     'Set right margin to 1/2

End Sub
```

MarginTop Property

Applies To

ssPrintInfo Object

Description

Determines the top margin of a printed page of data.

Syntax

object.**MarginTop**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	A single precision expression specifying the top margin of the printout.

Remarks

The **MarginTop** property specifies the distance between the top of the printed text and the top of the page on a printout. This includes any text specified for the page header.

The system of measurement used by the operating system (as specified by Regional Settings Properties in the Control Panel) will determine the units used by the **MarginTop** property. If the U.S. (English) system is being used, the setting will be in inches. If the Metric system is being used, the setting will be in millimeters.

See Also

MarginBottom, MarginLeft, MarginRight

Example

The **PrintBegin** event fires just before the beginning of a print operation. You can use it to set various print options. Note that if a Print or Print Setup dialog was presented to the user, they have already had the opportunity to set these options, in which case the following code is not necessary unless you want to override one or more of the settings.

In this example, you explicitly set all of the print options.

If the printout is too wide to fit on one page, **PageBreakOnGroups** controls how to split the grid over multiple pages. Here you specify that the split should occur on group boundaries and that pages should be left-aligned. (Other options are to break on columns or to break on groups but center the data on the page).

```
Private Sub SSDBGrid1_PrintBegin(ByVal ssPrintInfo As ssPrintInfo)

   ssPrintInfo.Copies = 5             'Print 5 copies.
   ssPrintInfo.Collate = True         'The copies should be collated.
   ssPrintInfo.PageStart = 2          'Just print pages 2 through 5
   ssPrintInfo.PageEnd = 5
   ssPrintInfo.Portrait = True        'Print portrait style
                                      '(set to false for landscape)
```

```
    ssPrintInfo.PageBreakOnGroups = ssPrintLeftAlignGroups

    'Set margins.
    'If the U.S. (English) system is being used,
    'the setting will be in inches.
    'If the Metric system is being used,
    'the setting will be in millimeters.
    ssPrintInfo.MarginBottom = 1     'Set bottom margin to 1
    ssPrintInfo.MarginTop = 1        'Set top margin to 1
    ssPrintInfo.MarginLeft = 0.5     'Set left margin to 1/2
    ssPrintInfo.MarginRight = 0.5    'Set right margin to 1/2

End Sub
```

Mask Property

Applies To

Column object, SSDBCombo

Description

Returns or sets a value that determines the input mask for the object.

Syntax

object.**Mask**[= string]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
string	A string expression that will be used to mask data. If data entered does not match the mask, a **ValidationError** event occurs.

Remarks

The **Mask** property provides data input masking features for any column in the SSDBGrid or the SSDBCombo. If data that is entered does not fit the specified mask, a **ValidationError** event occurs.

When the value of the **Mask** property is an empty string (""), the column or control will not mask entered or retrieved data. When an input mask is defined, underscores appear beneath every placeholder in the mask. You can only replace a placeholder with a character that is of the same type as the one specified in the input mask. If you enter an invalid character, the control rejects the character and generates a **ValidationError** event. Masking is provided primarily to data input, although in some cases the mask will be used to format text in a bound control that is read in from the database.

Multiline editing is not supported when using a mask. If you set the **Mask** property of an object to something other than an empty string, multiline editing will be disabled for that object.

For example, if the data mask of an object such as a Data Grid column is set up for Social Security number entry (`###-##-####`) and a purely numeric value is read into the column from the database (`111223333`) the mask will format the value and the number will be displayed as `111-22-3333`. However if you are using the same mask and an invalid value is read in from the database (`1A4ZX23Q8`) the **ValidationError** event will not fire. Deleting part of the string will not cause the prompt characters to appear, although if you attempt to enter more invalid data, it will be masked and the **ValidationEvent** will fire. If you delete the contents of the cell, the mask will be refreshed and the prompt characters will reappear. Any further data entry you do will be subject to data masking and may fire the **ValidationError** event.

When the object loses focus, its contents may be modified depending on other parameters of the object. For example, if the object has a value specified for its **NumberFormat** property, and the data can be expressed in the specified format, the object's text will be replaced with the newly formatted data.

Data entry will always overwrite the current prompt character (usually the underscore _). Data entry generally takes place in overstrike (as opposed to insert) mode. If the cursor is between two valid input characters, any characters to the right of the inserted entry will be pushed further to the right, unless the new string created would not pass a validation check. In that case a **ValidationError** event occurs, and the new character is not inserted.

Note that although you can mask input data, the control will not perform any additional data parsing. Data is compared to the mask exactly as input, and any data that does not match the mask generates the **ValidationError** event. For example, suppose you have defined a mask of "99.99" and the user inputs the following data:

```
3_.55
```

In this case, the user entered 3, pressed the right arrow, then entered .55. The control will not interpret this entry as "3.55". It will instead generate a **ValidationError** event, because the space (underscore) character does not match the mask.

Note If you are using a date-entry mask (such as ##/##/####) for an object bound to a date field in the database, you must be sure to include the prompt characters as part of the data by setting the **PromptInclude** property to True. Otherwise, the data will be entered in the database in purely numeric format, generating a data conversion error.

You can define input masks at both design time and run time. However, the following are examples of standard input masks that you may want to use at design time. The control can distinguish between numeric and alphabetic characters for validation, but cannot check for valid content, such as the correct month or time of day.

Mask	Description
Null String	(Default) No mask. Acts like a standard text box.
##-???-##	Medium date (US). Example: 17-Feb-96
##-##-##	Short date (US). Example: 05-10-94
##:## ??	Medium time. Example: 06:48 AM
##:##	Short time. Example: 18:48

The input mask can consist of the following characters.

Mask character	Description
#	Digit placeholder.
.	Decimal placeholder. The actual character used is the one specified as the decimal placeholder in your international settings. This character is treated as a literal for masking purposes.
,	Thousands separator. The actual character used is the one specified as the thousands separator in your international settings. This character is treated as a literal for masking purposes.
:	Time separator. The actual character used is the one specified as the time separator in your international settings. This character is treated as a literal for masking purposes.
/	Date separator. The actual character used is the one specified as the date separator in your international settings. This character is treated as a literal for masking purposes.
\	Treat the next character in the mask string as a literal. This allows you to include the '#', '&', 'A', and '?' characters in the mask. This character is treated as a literal for masking purposes.
&	Character placeholder. Valid values for this placeholder are ANSI characters in the following ranges: 32-126 and 128-255.
>	Convert all the characters that follow to uppercase.
<	Convert all the characters that follow to lowercase.
A	Alphanumeric character placeholder (entry required). For example: a - z, A - Z, or 0 - 9.
a	Alphanumeric character placeholder (entry optional).
9	Digit placeholder (entry optional). For example: 0 - 9.

C	Character or space placeholder (entry optional). This operates exactly like the & placeholder, and ensures compatibility with Microsoft Access.
?	Letter placeholder. For example: a - z or A - Z.
Literal	All other symbols are displayed as literals; that is, as themselves.

Note When you define an input mask for a grid column, the **ValidationError** event is generated if there are any invalid characters in the active cell when it loses focus.

See Also

ClipMode, PromptChar, NumberFormat, PromptInclude, ValidationError

Example

This routine intializes a masked edit field in the first column of the Data Grid.

```
Sub InitializeMaskEdit

   'Set the mask in the first column
   SSDBGrid1.Columns(0).Mask = "######.##"

   'Use question mark as the prompt character
   SSDBGrid1.Columns(0).PromptChar = "?"

   'Specify that the prompt character not be included
   'in the returned string
   SSDBGrid1.Columns(0).PromptInclude = False

End Sub
```

MaxDropDownItems Property

Applies To

SSDBCombo, SSDBDropDown

Description

Sets or returns the maximum number of items visible in a dropdown at once.

Syntax

object . **MaxDropDownItems**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the maximum number of items visible in a dropdown at once.

See Also

MinDropDownItems

MaxLinesPerRow Property

Applies To

ssPrintInfo Object

Description

Determines the maximum number of lines in an auto-sized row.

Syntax

object.**MaxLinesPerRow**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the maximum number of lines of text that can appear in a row when **RowAutoSize** is set to True.

Remarks

The **MaxLinesPerRow** property works in conjunction with the **RowAutoSize** property. If **RowAutoSize** is set to False, this property has no effect.

If **RowAutoSize** is set to True, this property indicates the maximum number of lines a row will extend to accomodate a long text field. Normally, an autosized row will extend its height to accomodate all the text in a long text field, until it reaches the bottom of the page. By setting **MaxLinesPerRow** to a non-zero value, you can limit the number of lines of text that will be printed in any given row of the report. (Text is always wrapped within the width of the column containing the text field, as determined by the grid layout.) Text that does not fit within the specified number of lines will be cropped.

Setting **MaxLinesPerRow** to zero (the default value) disables the maximum setting. The row will be sized as large as necessary. You may also want to note that setting **MaxLinesPerRow** to one is effectively the same as setting **RowAutoSize** to False, except that it causes extra code to be executed.

See Also

RowAutoSize

Example

The **PrintInitialize** event fires before the beginning of a print job to give you a way to set up the values that will be be displayed to the user in the Print and Print Setup dialog boxes. The **PrintBegin** event fires just before printing actually begins (after any print dialogs have been displayed).

```
Private Sub SSDBGrid1_PrintInitialize(ByVal ssPrintInfo As ssPrintInfo)

   'Define a page header that includes a page number
   ssPrintInfo.PageHeader = "Print Sample: Page <page number>"
   'Define a page footer that specifies when the grid was printed.
   ssPrintInfo.PageFooter = "Printed <date> <time>"

   'Specify that we want each row's height to expand
   'so that all data is displayed,
   'but up to a maximum of 10 lines.
   ssPrintInfo.RowAutoSize = True      'So rows are expanded in height
                                       'as necessary
   ssPrintInfo.MaxLinesPerRow = 10     'but up to a maximum of 10
                                       'lines.
```

```
    'Indicate that we do not want to print colors.
    ' (Use True to print colors or shades
    'of gray on a non-color printer).
    ssPrintInfo.PrintColors = False

    'Print the column headers.
    ssPrintInfo.PrintColumnHeaders = ssUseCaption

    'The only grid lines we want to print are to separate groups.
    '(There are other options to print various types of grid lines.)
    ssPrintInfo.PrintGridLines = ssPrintGridLinesColGrpHeadsOnly

    'Print column and group headers at the top of each page.
    '(Use ssTopOfReport if you want the headers to appear on the
    'first page.)
    ssPrintInfo.PrintHeaders = ssTopOfPage

End Sub
```

MaxSelectedRows Property

Applies To

SSDBGrid

Description

Sets or returns the maximum number of rows that can be selected at any one time.

Syntax

object . **MaxSelectedRows**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

See Also

SelChange event, SelectTypeRow property

MinColWidth Property

Applies To

SSDBOptSet

Description

Sets or returns the minimum column width for the column containing the selected button.

Syntax

object . **MinColWidth**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

Example

The following example sets the minimum column width for the fourth button to 50:

```
SSDBOptSet1.Buttons(3).MinColWidth = 50
```

MinDropDownItems Property

Applies To

SSDBCombo, SSDBDropDown

Description

Sets or returns the minimum number of items visible in a dropdown at once.

Syntax

object . **MinDropDownItems**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the minimum number of items visible in a dropdown at once.

See Also

MaxDropDownItems

MinHeight Property

Applies To

SSDBOptSet

Description

Sets or returns the minimum height of the control.

Syntax

object . **MinHeight**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

Example

The following example sets the minimum height of the control to 250:

```
SSDBOptSet1.MinHeight = 250
```

MouseIcon Property

Applies To

SSDBCombo, SSDBCommand, SSDBData, SSDBOptSet

Description

Sets the custom icon to be used when the mouse passes over the control.

Syntax

object . **MouseIcon**[= *picture*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
picture	A string expression specifying the name of a file containing a graphic, as described in Settings.

Settings

Setting	Description
(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.

See Also

MousePointer

MousePointer Property

Applies To

SSDBCombo, SSDBCommand, SSDBData, SSDBOptSet

Description

Determines the icon displayed when the mouse passes over the control.

Syntax

object . **MousePointer**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

See Also

MouseIcon

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
```

MoveFirst Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Moves to the first record in the grid.

Syntax

object . **MoveFirst**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.

See Also

MoveLast, MoveNext, MovePrevious, MoveRecords

MoveLast Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Moves to the last record in the grid.

Syntax

object . **MoveLast**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.

See Also

MoveFirst, MoveNext, MovePrevious, MoveRecords

MoveNext Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Moves to the next record in the grid.

Syntax

object . **MoveNext**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.

See Also

MoveFirst, MoveNext, MovePrevious, MoveRecords

MovePrevious Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Moves to the previous record in the grid.

Syntax

object . **MovePrevious**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.

See Also

MoveFirst, MoveLast, MoveNext, MoveRecords

MoveRecords Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Moves a specified number of records in the grid.

Syntax

object . **MoveRecords(** *Records* **As Long)**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
Records	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.

See Also

MoveFirst, MoveLast, MoveNext, MovePrevious

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)
```

MultiLine Property

Applies To

SSDBCombo, SSDBGrid

Description

Determines whether the contol will display multiple lines of text.

Syntax

object . **MultiLine**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression how text will be displayed, as described in Settings.

Settings

Setting	Description
True	(Default) The object will display multiple lines of text.
False	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.

This property also determines if a carriage return will be added to the text of a cell when the ENTER key is pressed. If set to **False**, a beep is sounded when the ENTER key is pressed.

Note Typing multiple lines does not increase the size of the cell or SSDBCombo.

See Also

VertScrollBar, WordWrap

Name Property

Applies To

Font object, Headfont object, StyleSet object

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**[= *styleset*]

object .**Name**[= *string*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
font	A string expression specifying the name of the font to use.
styleset	A string expression specifying the name of the styleset to use.
string	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.

Nullable Property

Applies To

Column object

Description

Determines how the control stores null or empty data in the database (for the Text data type only)

Syntax

object . **Nullable**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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 ("").
3	Empty Variant. The data will be stored as an empty variant. Used because SQL server recognizes empty strings as a null values.

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.

NumberFormat Property

Applies To

Column object, SSMaskedEdit control

Description

Sets or returns the display format for the object.

Syntax

object . **NumberFormat**[= *format*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
format	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.

NumberFormat is only applied when a cell does not have focus. When a cell has focus, it is displayed as it is stored in the underlying data.

Changing the format of a cell has no affect on its actual data, only the way that data appears when the cell does not have focus.

Note This property only functions in Visual Basic, and is not supported in other environments, such as Visual C++ and Internet Explorer.

The following is a list of the named format strings that can be used with **NumberFormat**:

Format name	Description
General Number	Display number as is, with no thousand separators.
Currency	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.
Fixed	Display at least one digit to the left and two digits to the right of the decimal separator.
Standard	Display number with thousands separator, at least one digit to the left and two digits to the right of the decimal separator.
Percent	Display number multiplied by 100 with a percent sign (%) appended to the right; always display two digits to the right of the decimal separator.
Scientific	Use standard scientific notation.
Yes/No	Display No if number is 0; otherwise, display Yes.
True/False	Display False if number is 0; otherwise, display True.
On/Off	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
None	**No formatting**. Display the number with no formatting.
0	**Digit placeholder**. 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).

See Also

Visual Basic's **Format** Function, Mask property

Example

The following examples demonstrate the effects **NumberFormat** has on data:

Data	NumberFormat String	Displayed Result
5459.4	$ ##,##0.00	$ 5,459.40
334.9	###0.000	334.900
100	Currency	$100.00

NumberOfButtons Property

Applies To

SSDBOptSet

Description

Sets or returns the number of option buttons in the DataOptionSet.

Syntax

object . **NumberOfButtons**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

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
```

See Also

IndexSelected

OptionValue Property

Applies To

Button object, SSDBOptSet

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*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
Text	A string expression that evaluates to the value of a field.

Remarks

Each button must have unique option values.

See Also

DataField, DataSource

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"
```

Orientation Property

Applies To

SSDBData

Description

Sets or returns the control's display orientation, either horizontal or vertical.

Syntax

object . **Orientation**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the display orientation of the control, as described in Settings.

Settings

Setting	Description
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.

PageBreakOnGroups Property

Applies To

ssPrintInfo Object

Description

Determines how reports that are too wide to fit on one page will span multiple pages.

Syntax

*object.***PageBreakOnGroups**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying how groups will break across pages, as described in Settings.

Settings

Setting	Description
0	(Default) *ssPrintPageBreakOnColumns*. Pages will be broken based on columns. Groups will be split if necessary.
1	*ssPrintLeftAlignGroups*. Pages will be broken based on group boundaries. Groups printed on the page will be left aligned.
2	*ssPrintCenterGroups*. Pages will be broken based on group boundaries. Groups printed on the page will be centered horizontally.

There are constants available for the settings of this property.

Remarks

The **PageBreakOnGroups** property determines how reports that are too wide to fit on a single page will be broken. The default behavior is to break on a column. If a column cannot fit on a page given the current margin settings, it will be printed on the next page. If the column is a member of a group, that group will be broken, printing on two or more pages until all the columns it contains are printed.

You may want to keep groups together on a single page. You can do this by changing the setting of **PageBreakOnGroups**. When set to *ssPrintLeftAlignGroups* (1) or *ssPrintCenterGroups* (2) the control will break pages on group boundaries only. If a group is too large to fit on a page, it will be moved to the next page. This

may leave a large amount of blank space on any given page, so you can select whether the groups that do fit will be centered or left aligned.

If you have a group that is too wide to fit on a single page and you have specified that pages should break on group boundaries, that group will be scaled horizontally in order to fit on a single page. The columns contained by the group will be made proportionally narrower so that the entire group fits.

Example

The **PrintBegin** event fires just before the beginning of a print operation. You can use it to set various print options. Note that if a Print or Print Setup dialog was presented to the user, they have already had the opportunity to set these options, in which case the following code is not necessary unless you want to override one or more of the settings.

In this example, you explicitly set all of the print options.

If the printout is too wide to fit on one page, **PageBreakOnGroups** controls how to split the grid over multiple pages. Here you specify that the split should occur on group boundaries and that pages should be left-aligned. (Other options are to break on columns or to break on groups but center the data on the page).

```
Private Sub SSDBGrid1_PrintBegin(ByVal ssPrintInfo As ssPrintInfo)

   ssPrintInfo.Copies = 5           'Print 5 copies.
   ssPrintInfo.Collate = True       'The copies should be collated.
   ssPrintInfo.PageStart = 2        'Just print pages 2 through 5
   ssPrintInfo.PageEnd = 5
   ssPrintInfo.Portrait = True      'Print portrait style
                                    '(set to false for landscape)

   ssPrintInfo.PageBreakOnGroups = ssPrintLeftAlignGroups

   'Set margins.
   'If the U.S. (English) system is being used,
   'the setting will be in inches.
   'If the Metric system is being used,
   'the setting will be in millimeters.
   ssPrintInfo.MarginBottom = 1     'Set bottom margin to 1
   ssPrintInfo.MarginTop = 1        'Set top margin to 1
   ssPrintInfo.MarginLeft = 0.5     'Set left margin to 1/2
   ssPrintInfo.MarginRight = 0.5    'Set right margin to 1/2

End Sub
```

PageEnd Property

Applies To

ssPrintInfo Object

Description

Determines the last page that will be printed.

Syntax

object.**PageEnd**[= number]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.

number An integer expression specifying the final page to be printed.

Remarks

This property, along with **PageStart**, gives you the ability to specify a range of pages to be printed. Printing will begin with the page specified by **PageStart** and finish with the page specified by **PageEnd**. For example, if you have a ten page report, but you only wish to print the first five pages, you would specify a value of 1 for **PageStart** and a value of 5 for **PageEnd**.

Valid values for **PageEnd** are from 1 to 65535. Setting the value of this property to zero produces an error.

If you specify a value for **PageEnd** but not for **PageStart**, **PageStart** will be set to the same value you specified for **PageEnd**.

See Also

PageStart

Example

The **PrintBegin** event fires just before the beginning of a print operation. You can use it to set various print options. Note that if a Print or Print Setup dialog was presented to the user, they have already had the opportunity to set these options, in which case the following code is not necessary unless you want to override one or more of the settings.

In this example, you explicitly set all of the print options.

If the printout is too wide to fit on one page, **PageBreakOnGroups** controls how to split the grid over multiple pages. Here you specify that the split should occur on group boundaries and that pages should be left-aligned. (Other options are to break on columns or to break on groups but center the data on the page).

```
Private Sub SSDBGrid1_PrintBegin(ByVal ssPrintInfo As ssPrintInfo)

   ssPrintInfo.Copies = 5           'Print 5 copies.
   ssPrintInfo.Collate = True       'The copies should be collated.
   ssPrintInfo.PageStart = 2        'Just print pages 2 through 5
   ssPrintInfo.PageEnd = 5
   ssPrintInfo.Portrait = True      'Print portrait style
                                    '(set to false for landscape)

   ssPrintInfo.PageBreakOnGroups = ssPrintLeftAlignGroups

   'Set margins.
   'If the U.S. (English) system is being used,
   'the setting will be in inches.
   'If the Metric system is being used,
   'the setting will be in millimeters.
   ssPrintInfo.MarginBottom = 1     'Set bottom margin to 1
   ssPrintInfo.MarginTop = 1        'Set top margin to 1
   ssPrintInfo.MarginLeft = 0.5     'Set left margin to 1/2
   ssPrintInfo.MarginRight = 0.5    'Set right margin to 1/2

End Sub
```

PageFooter Property

Applies To

ssPrintInfo Object

Description

Specifies the text that will be printed at the bottom of each page.

Syntax

object.**PageFooter**[= *text*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
text	A string expression that evaluates to the text of the page footer

Remarks

You can justify individual sections of the page footer by specifying a tab-delimited string for the **PageFooter** property. Text specified will be left-aligned until a tab character is encountered. Text following the first tab character that comes before the second tab character will be centered. Text following the second tab character will be right-aligned. For example, you could right align the entire page footer by beginning the text string with two tab characters.

The **PageFooter** property also recognizes field codes for the page number, date and time. If these are inserted into the text of the property, they will be replaced with the correct values when the report is printed. The date and time information used is always the current date and time when the print job begins.

The **PageFooter** property makes use of the following fields. Enter the field as shown with angle brackets (< >) in the text string you specify for the property:

Data Field	Value
`<page number>`	Replaced with the number of the current page
`<date>`	Replaced with the date the print job was started
`<time>`	Replaced with the time the print job was started

See Also

PageFooterFont, PageHeader

Example

The **PrintInitialize** event fires before the beginning of a print job to give you a way to set up the values that will be be displayed to the user in the Print and Print Setup dialog boxes. The **PrintBegin** event fires just before printing actually begins (after any print dialogs have been displayed).

```
Private Sub SSDBGrid1_PrintInitialize(ByVal ssPrintInfo As ssPrintInfo)

   'Define a page header that includes a page number
   ssPrintInfo.PageHeader = "Print Sample: Page <page number>"
   'Define a page footer that specifies when the grid was printed.
   ssPrintInfo.PageFooter = "Printed <date> <time>"

   'Specify that we want each row's height
   'to expand so that all data is displayed,
   'but up to a maximum of 10 lines.
   ssPrintInfo.RowAutoSize = True      'So rows are expanded in height
                                       'as necessary
   ssPrintInfo.MaxLinesPerRow = 10     'but up to a maximum of 10
                                       'lines.
```

```
    'Indicate that we do not want to print colors.
    '(Use True to print colors or shades
    'of gray on a non-color printer).
    ssPrintInfo.PrintColors = False

    'Print the column headers.
    ssPrintInfo.PrintColumnHeaders = ssUseCaption

    'The only grid lines we want to print are to separate groups.
    '(There are other options to print various types of grid lines.)
    ssPrintInfo.PrintGridLines = ssPrintGridLinesColGrpHeadsOnly

    'Print column and group headers at the top of each page.
    '(Use ssTopOfReport if you want the headers
    'to appear on the first page.)
    ssPrintInfo.PrintHeaders = ssTopOfPage

End Sub
```

PageFooterFont Property

Applies To

SSDBGrid

Description

Determines which font will be used for the page footer of a printout.

Syntax

object.**PageFooterFont**[= *font*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
font	A string expression that evaluates to the **Font** object that will be used to print the footer text.

Remarks

You can use the **PageFooterFont** property to specify the attributes of the page footer text. Because this property returns a reference to a **Font** object, you can use it to specify any property of that object. For example, to make the footer font bold, you would use the following code:

```
SSDBGrid1.PageFooterFont.Bold = True
```

See Also

PageFooter, PageHeaderFont

Example

This code is called when the form's "Print" button is clicked. It sets footer and header fonts then prints print the grid twice - once to print just selected rows (also specifying that the Print Setup dialog not be displayed but the Print dialog should be displayed) then again to print the entire grid with no print dialogs.

Note that other parameters can be set in the **PrintInitialize** event

```
Private Sub cmdPrint_Click()

   SSDBGrid1.PageFooterFont = "Times New Roman" 'Set page footer font.
   SSDBGrid1.PageHeaderFont = "Arial"           'Set page header font.
   'Print selected rows of the grid
   SSDBGrid1.PrintData ssPrintSelectedRows, False, True
   'Print all rows (no print dialogs)
   SSDBGrid1.PrintData ssPrintAllRows, False, False

End Sub
```

PageHeader Property

Applies To

ssPrintInfo Object

Description

Specifies the text that will be printed at the top of each page.

Syntax

object.**PageHeader**[= *text*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
text	A string expression that evaluates to the text of the page header.

Remarks

You can justify individual sections of the page header by specifying a tab-delimited string for the **PageHeader** property. Text specified will be left-aligned until a tab character is encountered. Text following the first tab character that comes before the second tab character will be centered. Text following the second tab character will be right-aligned. For example, you could right align the entire page header by beginning the text string with two tab characters.

The **PageHeader** property also recognizes field codes for the page number, date and time. If these are inserted into the text of the property, they will be replaced with the correct values when the report is printed. The date and time information used is always the current date and time when the print job begins.

The **PageHeader** property makes use of the following fields. Enter the field as shown with angle brackets (< >) in the text string you specify for the property:

Data Field	Value
`<page number>`	Replaced with the number of the current page.
`<date>`	Replaced with the date the print job was started.
`<time>`	Replaced with the time the print job was started.

See Also

PageFooter, PageHeaderFont

Example

The **PrintInitialize** event fires before the beginning of a print job to give you a way to set up the values that will

be be displayed to the user in the Print and Print Setup dialog boxes. The **PrintBegin** event fires just before printing actually begins (after any print dialogs have been displayed).

```
Private Sub SSDBGrid1_PrintInitialize(ByVal ssPrintInfo As ssPrintInfo)

   'Define a page header that includes a page number
   ssPrintInfo.PageHeader = "Print Sample: Page <page number>"
   'Define a page footer that specifies when the grid was printed.
   ssPrintInfo.PageFooter = "Printed <date> <time>"

   'Specify that we want each row's height
   'to expand so that all data is displayed,
   'but up to a maximum of 10 lines.
   ssPrintInfo.RowAutoSize = True      'So rows are expanded in height
                                       'as necessary
   ssPrintInfo.MaxLinesPerRow = 10     'but up to a maximum of 10
                                       'lines.

   'Indicate that we do not want to print colors.
   '(Use True to print colors or shades
   'of gray on a non-color printer).
   ssPrintInfo.PrintColors = False

   'Print the column headers.
   ssPrintInfo.PrintColumnHeaders = ssUseCaption

   'The only grid lines we want to print are to separate groups.
   '(There are other options to print various types of grid lines.)
   ssPrintInfo.PrintGridLines = ssPrintGridLinesColGrpHeadsOnly

   'Print column and group headers at the top of each page.
   '(Use ssTopOfReport if you want the headers
   'to appear on the first page.)
   ssPrintInfo.PrintHeaders = ssTopOfPage

End Sub
```

PageHeaderFont Property

Applies To

SSDBGrid

Description

Determines which font will be used for the page header of a printout.

Syntax

object.**PageHeaderFont**[= *font*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
font	A string expression that evaluates to the **Font** object that will be used to print the header text.

Remarks

You can use the **PageHeaderFont** property to specify the attributes of the page header text. Because this property returns a reference to a **Font** object, you can use it to specify any property of that object. For example, to make the header font bold, you would use the following code:

```
SSDBGrid1.PageFooterFont.Bold = True
```

See Also

PageFooterFont, PageHeader

Example

This code is called when the form's "Print" button is clicked. It sets footer and header fonts then prints print the grid twice - once to print just selected rows (also specifying that the Print Setup dialog not be displayed but the Print dialog should be displayed) then again to print the entire grid with no print dialogs.

Note that other parameters can be set in the **PrintInitialize** event

```
Private Sub cmdPrint_Click()

   SSDBGrid1.PageFooterFont = "Times New Roman" 'Set page footer font.
   SSDBGrid1.PageHeaderFont = "Arial"           'Set page header font.
   'Print selected rows of the grid
   SSDBGrid1.PrintData ssPrintSelectedRows, False, True
   'Print all rows (no print dialogs)
   SSDBGrid1.PrintData ssPrintAllRows, False, False

End Sub
```

PageStart Property

Applies To

ssPrintInfo Object

Description

Determines the first page that will be printed.

Syntax

object.**PageStart**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the first page to be printed.

Remarks

This property, along with **PageEnd**, gives you the ability to specify a range of pages to be printed. Printing will begin with the page specified by **PageStart** and finish with the page specified by **PageEnd**. For example, if you have a ten page report, but you only wish to print the last five pages, you would specify a value of 6 for **PageStart** and a value of 10 for **PageEnd**.

Valid values for **PageStart** are from 1 to 65535. Setting the value of this property to zero produces an error.

If you specify a value for **PageStart** but not for **PageEnd**, **PageEnd** will be set to the same value you specified for **PageStart**.

See Also

PageEnd

Example

The **PrintBegin** event fires just before the beginning of a print operation. You can use it to set various print options. Note that if a Print or Print Setup dialog was presented to the user, they have already had the opportunity to set these options, in which case the following code is not necessary unless you want to override one or more of the settings.

In this example, you explicitly set all of the print options.

If the printout is too wide to fit on one page, **PageBreakOnGroups** controls how to split the grid over multiple pages. Here you specify that the split should occur on group boundaries and that pages should be left-aligned. (Other options are to break on columns or to break on groups but center the data on the page).

```
Private Sub SSDBGrid1_PrintBegin(ByVal ssPrintInfo As ssPrintInfo)

   ssPrintInfo.Copies = 5           'Print 5 copies.
   ssPrintInfo.Collate = True       'The copies should be collated.
   ssPrintInfo.PageStart = 2        'Just print pages 2 through 5
   ssPrintInfo.PageEnd = 5
   ssPrintInfo.Portrait = True      'Print portrait style
                                    '(set to false for landscape)

   ssPrintInfo.PageBreakOnGroups = ssPrintLeftAlignGroups

   'Set margins.
   'If the U.S. (English) system is being used,
   'the setting will be in inches.
   'If the Metric system is being used,
   'the setting will be in millimeters.
   ssPrintInfo.MarginBottom = 1     'Set bottom margin to 1
   ssPrintInfo.MarginTop = 1        'Set top margin to 1
   ssPrintInfo.MarginLeft = 0.5     'Set left margin to 1/2
   ssPrintInfo.MarginRight = 0.5    'Set right margin to 1/2

End Sub
```

PageValue Property

Applies To

SSDBCommand, SSDBData

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*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the number of rows.

Remarks

The valid range for this property is 2-1000. The default value is 20.

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

Applies To

Button object, SSDBCommand, SSDBOptSet

Description

Determines a picture object for display.

Syntax

object . **Picture**[= *picture*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
picture	A string expression specifying the name of a file containing a graphic, as described in Settings.

Settings

Setting	Description
(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 **LoadPicture** 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.

See Also

AutoSize, PictureMetaHeight, PictureMetaWidth, PictureAlignment

Example

To load the picture at runtime you need to use the **LoadPicture** statement as shown here:

```
SSDBCommand1.Picture = LoadPicture("icons\writing\erase02.ico")
```

PictureAlignment Property

Applies To

SSDBCommand, SSDBOptSet

Description

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

Syntax

object . **PictureAlignment**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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

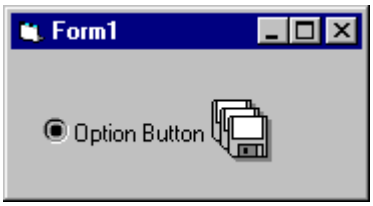
There are constants available for the settings of this property.

Remarks

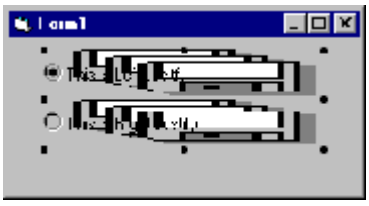
For the SSDBOptSet control, this property determines placement of the picture relative to the option button. This property is control-specific, and affects all buttons within the SSDBOptSet control.

The following examples demonstrate the various alignment positions as it applies to the SSDBOptSet control:

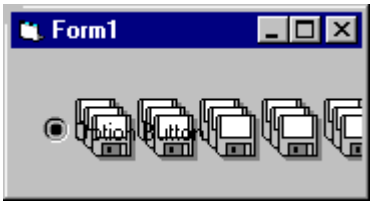
This is left of text



This is right of text



This is fit to caption



This is tiled

See Also

CaptionAlignment

PictureButton Property

Applies To

SSDBGrid

Description

Determines the picture to be used for the default button.

Syntax

object . **PictureButton**[= *picture*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
picture	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. You can load the graphic from the properties window at design time. At runtime, you can set this property using the **LoadPicture** function on a bitmap.

Remarks

This property will set the picture for the button (an ellipsis by default).

See Also

PictureComboButton, PictureRecordSelectors

PictureButtons Property

Applies To

SSDBData

Description

Determines the file containing the pictures to be used for each button in the control.

Syntax

object . **PictureButtons**[= *picture*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
Picture	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.

See Also

PictureCaption, PictureCaptionAlignment

Example

This sample code loads a custom **PictureButton** bitmap:

```
SSDBData.PictureButtons = LoadPicture ("c:\bitmaps\mybutns.bmp")
```

PictureCaption Property

Applies To

SSDBData

Description

Determines the picture to be drawn in the caption section of the control.

Syntax

object . **PictureCaption**[= *picture*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
picture	A string expression specifying the name of a file containing a graphic, as described in Settings.

Settings

Setting	Description
(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")
```

See Also

PictureButtons, PictureCaptionAlignment

PictureCaptionAlignment Property

Applies To

SSDBData

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*]

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

Settings

Setting	Description
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.

See Also

PictureButtons, PictureCaption

PictureCaptionMetaHeight Property

Applies To

SSDBData

Description

Sets or returns the height of a picture if it is a metafile.

Syntax

object . **PictureCaptionMetaHeight**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

PictureCaptionMetaWidth Property

Applies To

SSDBData

Description

Sets or returns the width of a picture if it is a metafile.

Syntax

object . **PictureCaptionMetaWidth**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

PictureComboButton Property

Applies To

SSDBGrid

Description

Determines the picture to be used for the dropdown button.

Syntax

object . **PictureComboButton**[= *picture*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
picture	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. You can load the graphic from the properties window at design time. At runtime, you can set this property using the **LoadPicture** function on a bitmap.

See Also

PictureButton, PictureRecordSelectors

PictureDropDown Property

Applies To

SSDBCombo

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*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
picture	An expression specifying a graphic, as described in Settings.

Settings

Setting	Description
(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 **LoadPicture** function.

PictureMetaHeight Property

Applies To

Button object, SSDBCommand, SSDBOptSet

Description

Sets or returns the height of a picture if it is a metafile.

Syntax

object . **PictureMetaHeight**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression that evaluates to the height of a metafile selected as a picture.

See Also

Picture, PictureMetaWidth

PictureMetaWidth Property

Applies To

Button object, SSDBCommand, SSDBOptSet

Description

Sets or returns the width of a picture if it is a metafile.

Syntax

object . **PictureMetaWidth**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression that evaluates to the width of a metafile selected as a picture.

See Also

Picture, PictureMetaHeight

PictureRecordSelectors Property

Applies To

SSDBGrid

Description

Determines the segmented bitmap for the record selectors graphic.

Syntax

object . **PictureRecordSelectors**[= *picture*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
picture	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. You can load the graphic from the properties window at design time. At runtime, you can set this property using the **LoadPicture** 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 automaticallly 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).

See Also

PictureButton, PictureComboButton

Portrait Property

Applies To

ssPrintInfo Object

Description

Determines the orientation of the printed page.

Syntax

object.**Portrait**[= boolean]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying page orientation, as described in Settings.

Settings

Setting	Description
True	(Default) The report will be printed on the page in portrait mode.
False	The report will be printed on the page in landscape mode.

Remarks

This setting determines the orientation of the printed page. You can choose to print in portrait (tall) or landscape (wide) mode. The capabilities of your print device may determine whether this setting has any effect.

Example

The **PrintBegin** event fires just before the beginning of a print operation. You can use it to set various print options. Note that if a Print or Print Setup dialog was presented to the user, they have already had the opportunity to set these options, in which case the following code is not necessary unless you want to override one or more of the settings.

In this example, you explicitly set all of the print options.

If the printout is too wide to fit on one page, **PageBreakOnGroups** controls how to split the grid over multiple pages. Here you specify that the split should occur on group boundaries and that pages should be left-aligned. (Other options are to break on columns or to break on groups but center the data on the page).

```
Private Sub SSDBGrid1_PrintBegin(ByVal ssPrintInfo As ssPrintInfo)

   ssPrintInfo.Copies = 5            'Print 5 copies.
   ssPrintInfo.Collate = True        'The copies should be collated.
   ssPrintInfo.PageStart = 2         'Just print pages 2 through 5
   ssPrintInfo.PageEnd = 5
   ssPrintInfo.Portrait = True       'Print portrait style
                                     '(set to false for landscape)

   ssPrintInfo.PageBreakOnGroups = ssPrintLeftAlignGroups

   'Set margins.
   'If the U.S. (English) system is being used,
   'the setting will be in inches.
   'If the Metric system is being used,
   'the setting will be in millimeters.
```

```
    ssPrintInfo.MarginBottom = 1      'Set bottom margin to 1
    ssPrintInfo.MarginTop = 1         'Set top margin to 1
    ssPrintInfo.MarginLeft = 0.5      'Set left margin to 1/2
    ssPrintInfo.MarginRight = 0.5     'Set right margin to 1/2

End Sub
```

Position Property

Applies To

Column object, Group object

Description

Sets or returns the position of the column or group within the grid.

Syntax

object . **Position**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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 or group does not reflect group assignments.

Position is a 0-based property indicating the ordinal number of the column or group in the control's current layout. For example, in a grid with 8 columns, the leftmost column's **Position** is zero, and the last column's **Position** is 7, whether or not that column is visible.

If a column is re-arranged within the control, either through code or via user interaction, its **Position** changes accordingly.

See Also

ColPosition method, GrpPosition method

PositionList Event

Applies To

SSDBCombo, SSDBDropDown

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**)

Part	Description
Text	The text to match in the list of values

Remarks

If the control has the **ListAutoPosition** property set to **False**, the control will generate this event when it needs to position the list, such as when the control is dropped down or when the user types while the dropdown portion is open. In this manner, 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.

See Also

ListAutoPosition

(DataCombo) Example

The following code shows one way you might position a bound DataCombo control in code, when ListAutoPosition is set to False.

In the SSDBCombo1_PositionList event:

```
Data1.Recordset.FindFirst "Author = '" & Text & "'"
If Data1.Recordset.NoMatch = False Then
    SSDBCombo1.Bookmark = Data1.Recordset.Bookmark
Else
    SSDBCombo1.MoveFirst
End If
```

The code above can be easily modified for use with an Unbound DataCombo, by performing the FindFirst on the recordset object, instead of the Data Control.

(DataDropDown) Example

The following code shows one way you might position a bound DataDropDown control in code, when ListAutoPosition is set to False.

In the SSDBDropDown1_PositionList event:

```
Data1.Recordset.FindFirst "Author = '" & Text & "'"
If Data1.Recordset.NoMatch = False Then
    SSDBDropDown1.Bookmark = Data1.Recordset.Bookmark
Else
    SSDBDropDown1.MoveFirst
End If
```

The code above can be easily modified for use with an Unbound DataDropDown, by performing the FindFirst on the recordset object, instead of the Data Control.

PrintBegin Event

Applies To

SSDBGrid

Description

Occurs at the beginning of a print job, when the job is about to be sent to the printer.

Syntax

Sub *control*_**PrintBegin** (*ssPrintInfo* **As ssPrintInfo**)

The event parameters are:

Parameter	Description
ssPrintInfo	An ssPrintInfo object that specifies the settings of the report about to be printed.

Remarks

The **PrintBegin** event occurs before the report is sent to the printer, but after the user has had the opportunity to configure the print job using the Print and Print Setup dialogs. The Print and Print Setup dialogs can be made available to the user when invoking the **PrintData** method, and will contain any default settings you have specified for the print job in the **PrintInitialize** event. The **PrintBegin** event is the last opportunity you have to change the parameters of a print job before it is committed to the print queue.

You can use the **PrintBegin** event to programmatically examine any changes to the **ssPrintInfo** object resulting from the user's actions. You can then choose to modify the user's settings where appropriate, or store them for later use.

The ssPrintInfo object is only accessible during this event and the **PrintInitialize** event.

See Also

PrintData, PrintError, PrintInitialize, RowPrint, ssPrintInfo

Example

The **PrintBegin** event fires just before the beginning of a print operation. You can use it to set various print options. Note that if a Print or Print Setup dialog was presented to the user, they have already had the opportunity to set these options, in which case the following code is not necessary unless you want to override one or more of the settings.

In this example, you explicitly set all of the print options.

If the printout is too wide to fit on one page, **PageBreakOnGroups** controls how to split the grid over multiple pages. Here you specify that the split should occur on group boundaries and that pages should be left-aligned. (Other options are to break on columns or to break on groups but center the data on the page).

```
Private Sub SSDBGrid1_PrintBegin(ByVal ssPrintInfo As ssPrintInfo)

   ssPrintInfo.Copies = 5           'Print 5 copies.
   ssPrintInfo.Collate = True       'The copies should be collated.
   ssPrintInfo.PageStart = 2        'Just print pages 2 through 5
   ssPrintInfo.PageEnd = 5
   ssPrintInfo.Portrait = True      'Print portrait style
                                    '(set to false for landscape)

   ssPrintInfo.PageBreakOnGroups = ssPrintLeftAlignGroups

   'Set margins.
   'If the U.S. (English) system is being used,
   'the setting will be in inches.
   'If the Metric system is being used,
   'the setting will be in millimeters.
   ssPrintInfo.MarginBottom = 1     'Set bottom margin to 1
   ssPrintInfo.MarginTop = 1        'Set top margin to 1
   ssPrintInfo.MarginLeft = 0.5     'Set left margin to 1/2
   ssPrintInfo.MarginRight = 0.5    'Set right margin to 1/2

End Sub
```

PrintColors Property

Applies To

ssPrintInfo Object

Description

Determines whether colors and shading will be printed.

Syntax

object.**PrintColors**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether colors and shading will be printed, as described in Settings.

Settings

Setting	Description
True	(Default) Colors specified in the grid layout will appear in the printed report, either as the specified color (color printers) or as a shade of gray (black and white printers).
False	Colors/shading will not be printed. Only data will appear in the printout.

Remarks

You can use the **PrintColors** property to determine whether a report will include the colors or shading in grid rows. Colors used will be those specified by the grid properties affecting color (i.e. BackColorEven, BackColorOdd) or by the stylesets applied to grid objects. Grid data will be printed in the colors specified on a color printer, or in a corresponding shade of gray on a monochrome printer. **PrintColors** also determines whether text will print in color.

If **PrintColors** is set to False, all output will be in black and white, with text appearing on a blank background.

See Also

PrintColumnHeaders, PrintGridLines, PrintGroupHeaders

Example

The **PrintInitialize** event fires before the beginning of a print job to give you a way to set up the values that will be be displayed to the user in the Print and Print Setup dialog boxes. The **PrintBegin** event fires just before printing actually begins (after any print dialogs have been displayed).

```
Private Sub SSDBGrid1_PrintInitialize(ByVal ssPrintInfo As ssPrintInfo)

   'Define a page header that includes a page number
   ssPrintInfo.PageHeader = "Print Sample: Page <page number>"
   'Define a page footer that specifies when the grid was printed.
   ssPrintInfo.PageFooter = "Printed <date> <time>"

   'Specify that we want each row's height
   'to expand so that all data is displayed,
   'but up to a maximum of 10 lines.
   ssPrintInfo.RowAutoSize = True      'So rows are expanded in height
                                       'as necessary
```

```
    ssPrintInfo.MaxLinesPerRow = 10      'but up to a maximum of 10
                                         'lines.

    'Indicate that we do not want to print colors.
    '(Use True to print colors or shades
    'of gray on a non-color printer).
    ssPrintInfo.PrintColors = False

    'Print the column headers.
    ssPrintInfo.PrintColumnHeaders = ssUseCaption

    'The only grid lines we want to print are to separate groups.
    '(There are other options to print various types of grid lines.)
    ssPrintInfo.PrintGridLines = ssPrintGridLinesColGrpHeadsOnly

    'Print column and group headers at the top of each page.
    '(Use ssTopOfReport if you want the headers
    'to appear on the first page.)
    ssPrintInfo.PrintHeaders = ssTopOfPage

End Sub
```

PrintColumnHeaders Property

Applies To

ssPrintInfo Object

Description

Determines whether columns will be printed with headers.

Syntax

object.**PrintColumnHeaders**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An enumerated integer expression specifying how column headers will appear in the printout, as described in Settings.

Settings

Setting	Description
0	*ssNoColumnHeaders*. Columns will be printed without headers.
1	(Default) *ssUseCaption*. The caption of the column will be used as the column header.
2	*ssUseFieldName*. The name of the column's data field will be used as the column header.

There are constants available for the settings of this property.

Remarks

The **PrintColumnHeaders** property determines how column headers will appear in the printout. Column headers appear at the top of each page, and serve to identify the contents of the columns of a report. You can choose to use the column captions from the grid layout as headers, use the name of the data fields to which the columns are bound, or forego column headers altogether.

You can determine whether column headers appear at the top of each page in the report or just once at the top of the report. The setting of the **PrintHeaders** property controls group and column header positioning. You can also toggle the display of group headers using the **PrintGroupHeaders** property.

See Also

PrintColors, PrintGridLines, PrintGroupHeaders, PrintHeaders

Example

The **PrintInitialize** event fires before the beginning of a print job to give you a way to set up the values that will be be displayed to the user in the Print and Print Setup dialog boxes. The **PrintBegin** event fires just before printing actually begins (after any print dialogs have been displayed).

```
Private Sub SSDBGrid1_PrintInitialize(ByVal ssPrintInfo As ssPrintInfo)

   'Define a page header that includes a page number
   ssPrintInfo.PageHeader = "Print Sample: Page <page number>"
   'Define a page footer that specifies when the grid was printed.
   ssPrintInfo.PageFooter = "Printed <date> <time>"

   'Specify that we want each row's height
   'to expand so that all data is displayed,
   'but up to a maximum of 10 lines.
   ssPrintInfo.RowAutoSize = True      'So rows are expanded in height
                                       'as necessary
   ssPrintInfo.MaxLinesPerRow = 10     'but up to a maximum of 10
                                       'lines.

   'Indicate that we do not want to print colors.
   '(Use True to print colors or shades
   'of gray on a non-color printer).
   ssPrintInfo.PrintColors = False

   'Print the column headers.
   ssPrintInfo.PrintColumnHeaders = ssUseCaption

   'The only grid lines we want to print are to separate groups.
   '(There are other options to print various types of grid lines.)
   ssPrintInfo.PrintGridLines = ssPrintGridLinesColGrpHeadsOnly

   'Print column and group headers at the top of each page.
   '(Use ssTopOfReport if you want the headers
   'to appear on the first page.)
   ssPrintInfo.PrintHeaders = ssTopOfPage

End Sub
```

PrintData Method

Applies To

SSDBGrid

Description

Prints the data from the grid using the current property settings of the ssPrintInfo object.

Syntax

object.**PrintData(**PrintDataFlags **As Integer**, ShowSetupDialog **As Boolean**, ShowPrintDialog **As Boolean**, Reserved **As Variant)**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
PrintDataFlags	An enumerated integer expression specifying the data to print, as described in Settings.
ShowSetupDialog	A Boolean expression that determines whether to display the Print Setup dialog before printing. Has no effect on 16-bit platforms.
ShowPrintDialog	A Boolean expression that determines whether to display the Print dialog before printing. Has no effect on 16-bit platforms.
Reserved	A variant expression. Unused in this version of Data Widgets.

Settings

Settings	Description
1	*ssPrintCurrentRow*. Prints only the row that is currently active (contains the active cell.)
2	*ssPrintSelectedRows*. Prints all selected rows.
4	*ssPrintNonSelectedRows*. Prints all rows that are not selected.
7	*ssPrintAllRows*. Prints all rows (Combines 1, 2 & 4)
16	*ssPrintFieldOrder*. Prints fields in straight field order, ignoring the grid layout. This flag must be combined with one or more of the other flags - it will not produce any output by itself.

There are constants available for the settings of this property.

Remarks

The **PrintData** method initiates a print job. It creates an ssPrintInfo object that contains information about the print job, such as how many copies will be printed, what the margins and layout of the page will be, the text of headers and footers, et cetera. When you invoke the **PrintData** method, you specify the data you want to be printed, and a number of other print options using the print data flags. You can also choose to display a standard windows Print Setup and/or and Print dialog to the user, allowing them to configure their own print settings.

Once you invoke the **PrintData** method, a **PrintInitialize** event occurs. the newly created ssPrintInfo object is passed to the **PrintInitialize** event, where you can set any of its properties to the default attributes of the print job. After the Print Setup and Print dialogs have been displayed, a **PrintBegin** event occurs, during which you can examine the properties of the ssPrintInfo object to see what changes the user has made to the print job.

The *PrintDataFlags* parameter represents a series of bit flags that specify what will be included in the print job and how the data should be arranged. Flags can be combined using the **Or** operator to specify multiple settings. Note that one of the first four flags is required to produce output; specifying *ssPrintFieldOrder* by itself will produce a blank page.

Note The DataGrid will *not* print graphics that are included as part of a styleset. Pictures in grid cells or captions will not appear when the data is printed.

See Also

PrintBegin, PrintError, PrintInitialize, RowPrint, ssPrintInfo

Example

This code is called when the form's "Print" button is clicked. It sets footer and header fonts then prints print the grid twice - once to print just selected rows (also specifying that the Print Setup dialog not be displayed but the Print dialog should be displayed) then again to print the entire grid with no print dialogs.

Note that other parameters can be set in the **PrintInitialize** event

```
Private Sub cmdPrint_Click()

   SSDBGrid1.PageFooterFont = "Times New Roman" 'Set page footer font.
   SSDBGrid1.PageHeaderFont = "Arial"           'Set page header font.
   'Print selected rows of the grid
   SSDBGrid1.PrintData ssPrintSelectedRows, False, True
   'Print all rows (no print dialogs)
   SSDBGrid1.PrintData ssPrintAllRows, False, False

End Sub
```

PrinterDeviceName Property

Applies To

ssPrintInfo Object

Description

Returns the name of the printer being used from the device driver.

Syntax

object . **PrinterDeviceName** = *name*

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
name	A string expression that evaluates to the name of the printer being used.

Remarks

Sheridan Software has tested Data Widgets 3.1's printing capabilities with an array of hardware devices and device drivers. We have found inconsistencies in the way certain printer drivers implement the printing APIs that Data Widgets uses to create its reports. Data Widgets 3.1 can detect the presence of these drivers and compensate automatically.

The **PrinterDeviceName** property is provided so that you can determine the name of the printing device being used, and take action based on the information. The name returned is the permanent device name specified by the printer driver, not the Windows printer name, which is user-configurable.

Note The **PrinterDeviceName** property is not available in the **PrintInitialize** event. If you check it's value in that event, it will be empty. Device information does not become available until the **PrintBegin** event. Any code that uses the **PrinterDeviceName** property should be placed in that event.

Depending on the printer being used, you may want to change or limit print settings. You can also perform print troubleshooting if needed. This property is useful when trying to determine whether to use the **ClippingOverride** property to deal with printing problems.

See Also

ClippingOverride, DriverOverride, PrinterDriverVer

PrinterDriverVer Property

Applies To

ssPrintInfo Object

Description

Returns the version number of the printer driver being used.

Syntax

object.**PrinterDriverVer**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the version of the printer driver.

Remarks

Sheridan Software has tested Data Widgets 3.1's printing capabilities with an array of hardware devices and device drivers. We have found inconsistencies in the way certain printer drivers implement the printing APIs that Data Widgets uses to create its reports. Data Widgets 3.1 can detect the presence of these drivers and compensate automatically.

Note The **PrinterDriverVer** property is not available in the **PrintInitialize** event. If you check its value in that event, it will be nothing. Device information does not become available until the **PrintBegin** event. Any code that uses the **PrinterDriverVer** property should be placed in that event.

The **PrinterDriverVer** property is provided so that you can determine the version of the printer driver being used to print the pending print job, and take action based on the information. This property is useful when trying to determine whether to use the **ClippingOverride** property to deal with printing problems.

See Also

ClippingOverride, DriverOverride, PrinterDeviceName

PrintError Event

Applies To

SSDBGrid

Description

Occurs in response to an error in the printing process.

Syntax

Sub *control*_**PrintError(***PrintError* **As Integer**, *Response* **As Integer)**

The event parameters are:

Parameter	Description
PrintError	An integer expression that specifies the error code identifying the error that occurred.
Response	An integer expression that specifies whether the error dialog will be shown, as specified in Settings.

Settings

The settings for *Response* are:

Value	Description
0	Continue.
1	(Default) Display the error message.

Remarks

The **PrintError** event gives you a way to respond to problems that may occur during the printing process.The **PrintError** parameter returns an error code (*PrintError*) you can examine to determine the cause of the error. You can also elect to allow the control to display an error dialog to the user, or you can suppress the error dialog display, optionally substituting your own. Setting the *Response* parameter to zero in the event code will suppress the display of the error dialog.

The following is a list of printing error codes and their meanings:

ErrorCode	Description
30454	"An internal printing error has occurred." Printing could not continue due to a problem with the program or the control, such as a data source failure.
30456	"Printing support file ssprn32.dll (or ssprn16.dll) missing or not registered." An essential 32-bit (or 16-bit) system file was not properly installed on the user's system.
30457	"Print job cancelled by user." The user manually stopped the print job before it was complete.
30458	"Printer was not available." Communication with the printer could not be established. Check that the printer is properly installed in Windows, has power, and is connected to the computer.

See Also

PrintBegin, PrintData, PrintInitialize, RowPrint, ssPrintInfo

Example

This event is fired when an error occurs during printing. In this example, you display your own error message and suppress the display of the default message. Nothing is displayed for error 30457, which just means that the user cancelled the print operation.

```
Private Sub SSDBGrid1_PrintError(ByVal PrintError As Long, Response As
Integer)

   If PrintError <> 30457 Then   'Ignore "cancelled by user" error
      MsgBox "A printing error occurred. " & _
      "Please see the network administrator. Error number = " _
      & Str$(PrintError), vbOKOnly, "Error!"
   End If

   'Set to 0 to prevent a default error message from being displayed
   Response = 0

End Sub
```

PrintGridLines Property

Applies To

ssPrintInfo Object

Description

Determines how grid lines will be printed.

Syntax

object.**PrintGridLines**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying where grid lines will be printed, as described in Settings.

Settings

Setting	Description
0	*ssPrintGridLinesNone*. Grid lines will not be printed. Only data will appear in the printout.
1	*ssPrintGridLinesColGrpHeadsOnly*. Only the grid lines that create borders around column and group headers will be printed
2	*ssPrintGridLinesRowsOnly*. Only the grid lines that create borders around data cells will be printed.
3	(Default) *ssPrintGridLinesAll*. All grid lines will be printed.

There are constants available for the settings of this property.

Remarks

You can use the **PrintGridLines** property to determine how grid lines will be printed in the report. Lines will be printed whether or not they are visible in the grid layout.

See Also

PrintColors, PrintColumnHeaders, PrintGroupHeaders

Example

The **PrintInitialize** event fires before the beginning of a print job to give you a way to set up the values that will be be displayed to the user in the Print and Print Setup dialog boxes. The **PrintBegin** event fires just before printing actually begins (after any print dialogs have been displayed).

```
Private Sub SSDBGrid1_PrintInitialize(ByVal ssPrintInfo As ssPrintInfo)

   'Define a page header that includes a page number
   ssPrintInfo.PageHeader = "Print Sample: Page <page number>"
   'Define a page footer that specifies when the grid was printed.
   ssPrintInfo.PageFooter = "Printed <date> <time>"

   'Specify that we want each row's height
   'to expand so that all data is displayed,
   'but up to a maximum of 10 lines.
   ssPrintInfo.RowAutoSize = True      'So rows are expanded in height
                                       'as necessary
   ssPrintInfo.MaxLinesPerRow = 10     'but up to a maximum of 10
                                       'lines.

   'Indicate that we do not want to print colors.
   '(Use True to print colors or shades
   'of gray on a non-color printer).
   ssPrintInfo.PrintColors = False

   'Print the column headers.
```

```
    ssPrintInfo.PrintColumnHeaders = ssUseCaption

    'The only grid lines we want to print are to separate groups.
    '(There are other options to print various types of grid lines.)
    ssPrintInfo.PrintGridLines = ssPrintGridLinesColGrpHeadsOnly

    'Print column and group headers at the top of each page.
    '(Use ssTopOfReport if you want the headers
    'to appear on the first page.)
    ssPrintInfo.PrintHeaders = ssTopOfPage

End Sub
```

PrintGroupHeaders Property

Applies To

ssPrintInfo Object

Description

Determines whether group headers will be printed.

Syntax

object.**PrintGroupHeaders**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether to print group headers, as described in Settings.

Settings

Setting	Description
True	(Default) Group headers will be included in the printout.
False	Group headers will not be printed.

Remarks

The **PrintGroupHeaders** property determines whether group headers will appear in the printout. Group headers appear above the column headers and serve to identify the column groupings in a report.

You can determine whether group headers appear at the top of each page in the report or just once at the top of the report. The setting of the **PrintHeaders** property controls group and column header positioning. You can also change the display of column headers using the **PrintColumnHeaders** property.

See Also

PrintColors, PrintColumnHeaders, PrintGridLines, PrintHeaders

PrintHeaders Property

Applies To

ssPrintInfo Object

Description

Determines where column and group headers will be printed.

Syntax

object.**PrintHeaders**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying where to print column and group headers, as described in Settings.

Settings

Setting	Description
0	*ssTopOfPage*. Column and group headers will appear at the top of every page.
1	(Default) *ssTopOfReport*. Column and group headers will appear only at the top of the first page.

There are constants available for the settings of this property.

Remarks

The **PrintHeaders** property controls the positioning of group and column headers in the report. Headers can appear once at the top of the report, or at the top of each page in the report.

You can also specify whether or not to display group and column headers, and what types of values column headers should display. Use the **PrintColumnHeaders** and **PrintGroupHeaders** properties to control these aspects of the printout.

See Also

PrintColumnHeaders, PrintGroupHeaders

Example

The **PrintInitialize** event fires before the beginning of a print job to give you a way to set up the values that will be be displayed to the user in the Print and Print Setup dialog boxes. The **PrintBegin** event fires just before printing actually begins (after any print dialogs have been displayed).

```
Private Sub SSDBGrid1_PrintInitialize(ByVal ssPrintInfo As ssPrintInfo)

   'Define a page header that includes a page number
   ssPrintInfo.PageHeader = "Print Sample: Page <page number>"
   'Define a page footer that specifies when the grid was printed.
   ssPrintInfo.PageFooter = "Printed <date> <time>"

   'Specify that we want each row's height
   'to expand so that all data is displayed,
   'but up to a maximum of 10 lines.
   ssPrintInfo.RowAutoSize = True      'So rows are expanded in height
                                       'as necessary
   ssPrintInfo.MaxLinesPerRow = 10     'but up to a maximum of 10
                                       'lines.

   'Indicate that we do not want to print colors.
   '(Use True to print colors or shades
```

```
    'of gray on a non-color printer).
    ssPrintInfo.PrintColors = False

    'Print the column headers.
    ssPrintInfo.PrintColumnHeaders = ssUseCaption

    'The only grid lines we want to print are to separate groups.
    '(There are other options to print various types of grid lines.)
    ssPrintInfo.PrintGridLines = ssPrintGridLinesColGrpHeadsOnly

    'Print column and group headers at the top of each page.
    '(Use ssTopOfReport if you want the headers
    'to appear on the first page.)
    ssPrintInfo.PrintHeaders = ssTopOfPage

End Sub
```

PrintInitialize Event

Applies To

SSDBGrid

Description

Occurs just before the beginning of a print job, when the job is first initiated.

Syntax

Sub *control*_**PrintInitialize** (*ssPrintInfo* **As ssPrintInfo**)

The event parameters are:

Parameter	Description
ssPrintInfo	An ssPrintInfo object that specifies the settings of the report about to be printed.

Remarks

The **PrintInitialize** event occurs when the print job is first initiated via the **PrintData** method. It gives you the opportunity to set the default parameters for the print job (number of copies, page orientation, header & footer text, etc.) After you have set up the default print settings in the **PrintInitialize** event, the Print Setup and Print dialogs may be displayed to the user, depending on the parameters passed to the **PrintData** method. The user can use these dialogs to change the defaults you have specified. Once the user has completed their changes, the **PrintBegin** event occurs, giving you the chance to examine their settings, change them if necessary or store them for future use.

The ssPrintInfo object is only accessible during this event and the **PrintBegin** event.

See Also

PrintBegin, PrintData, PrintError, RowPrint, ssPrintInfo

Example

The **PrintInitialize** event fires before the beginning of a print job to give you a way to set up the values that will be be displayed to the user in the Print and Print Setup dialog boxes. The **PrintBegin** event fires just before printing actually begins (after any print dialogs have been displayed).

```
Private Sub SSDBGrid1_PrintInitialize(ByVal ssPrintInfo As ssPrintInfo)
```

```
    'Define a page header that includes a page number
    ssPrintInfo.PageHeader = "Print Sample: Page <page number>"
    'Define a page footer that specifies when the grid was printed.
    ssPrintInfo.PageFooter = "Printed <date> <time>"

    'Specify that we want each row's height
    'to expand so that all data is displayed,
    'but up to a maximum of 10 lines.
    ssPrintInfo.RowAutoSize = True      'So rows are expanded in height
                                        'as necessary
    ssPrintInfo.MaxLinesPerRow = 10     'but up to a maximum of 10
                                        'lines.

    'Indicate that we do not want to print colors.
    '(Use True to print colors or shades
    'of gray on a non-color printer).
    ssPrintInfo.PrintColors = False

    'Print the column headers.
    ssPrintInfo.PrintColumnHeaders = ssUseCaption

    'The only grid lines we want to print are to separate groups.
    '(There are other options to print various types of grid lines.)
    ssPrintInfo.PrintGridLines = ssPrintGridLinesColGrpHeadsOnly

    'Print column and group headers at the top of each page.
    '(Use ssTopOfReport if you want the headers
    'to appear on the first page.)
    ssPrintInfo.PrintHeaders = ssTopOfPage

End Sub
```

PromptChar Property

Applies To

Column object, SSDBCombo

Description

Returns or sets a value that determines the prompt character used during masked input.

Syntax

object.**PromptChar**[= *string*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
string	A string expression that evaluates to the single character used to prompt the user for data. If more than one character is specified, only the first will be used.

Remarks

The default character for masked input is the underscore (_) character. You can use the **PromptChar** property to specify a different character if you want. Only a single character may be used as the prompt character.

See Also

ClipMode, Mask, PromptInclude, ValidationError

Example

This routine intializes a masked edit field in the first column of the Data Grid.

```
Sub InitializeMaskEdit

   'Set date mask in the first column
   SSDBGrid1.Columns(0).Mask = "######.##"

   'Use question mark as the prompt character
   SSDBGrid1.Columns(0).PromptChar = "?"

   'Specify that the prompt character not be included
   'in the returned string
   SSDBGrid1.Columns(0).PromptInclude = False

End Sub
```

PromptInclude Property

Applies To

SSDBCombo

Description

Determines whether literal characters are included in the object's **Text** property.

Syntax

object.**PromptInclude**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether to include mask prompt characters in the **Text** property of the object, as described in Settings.

Settings

Setting	Description
True	Literal characters will be included in the **Text** property of the object.
False	(Default) Prompt characters will not be included in the **Text** property of the object.

Remarks

The **PromptInclude** property only applies to objects that have data masking enabled (the **Mask** property is set to something other than an empty string.) The function of this property in Data Widgets varies slightly from its function in the Microsoft Masked Edit control.

If **PromptInclude** is set to True for an object that is bound to a text field in a database (as determined by the object's **DataField** property) then any prompt characters (as determined by the value of the **PromptChar** property) that appear in the text of the object will be written to the database as spaces. If **PromptInclude** is set to False, prompt characters will be removed altogether, and the data preceding and following the prompt character will be concatenated.

If the **PromptInclude** property is set to True, the literal characters displayed by the control (such as the "/" characters in a date field or the "$" character in a currency field) are considered part of the object's text and will be

returned as part of the **Text** property. Also, literal characters will be written to the database as part of the data when the control is bound to a data source.

When **PromptInclude** is set to False, literal characters are ignored. The object's **Text** property returns only the user's data, and no mask characters will be written to the database.

PromptInclude also affects to the **CellText** property, the **CellValue** property, and the **Value** property of the Column object, but only if the column's **DataType** property is set to 8 (Text). If the column is not a text column, the setting of **PromptInclude** has no effect.

Note If you are using a date-entry mask (such as ##/##/####) for an object bound to a date field in the database, you must be sure to include the prompt characters as part of the data by setting the **PromptInclude** property to True. Otherwise, the data will be entered in the database in purely numeric format, generating a data conversion error.

See Also

ClipMode, Mask, PromptChar, ValidationError

Example

This routine initializes a masked edit field in the first column of the Data Grid.

```
Sub InitializeMaskEdit

   'Set date mask in the first column
   SSDBGrid1.Columns(0).Mask = "######.##"

   'Use question mark as the prompt character
   SSDBGrid1.Columns(0).PromptChar = "?"

   'Specify that the prompt character not be included
   'in the returned string
   SSDBGrid1.Columns(0).PromptInclude = False

End Sub
```

ReadType Property

Applies To

ssRowBuffer

Description

Determines what data is needed by the RowBuffer object in the **UnboundReadData** event. The control automatically sets this property to indicate what type of data must be provided.

Syntax

object . **ReadType**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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, specified by the control's **DataFieldList** property. This setting does not apply to the SSDBGrid control.
3	Read all bookmarks and data for displayed column only, specified by the control's **DataFieldToDisplay** property. This setting does not apply to the SSDBGrid control.

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.

See Also

DataFieldToDisplay property, UnboundReadData event, Performance Tuning

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 each row in the row buffer
For i = 0 To RowBuf.RowCount - 1
   'If the pointer is outside the grid then stop this
   If p < 0 Or p > counter Then Exit For

      'Optimize the data read based on the ReadType
      Select Case RowBuf.ReadType
      Case 0   'Read all data
         'For each column in the grid
         For j = 0 To SSDBGrid1.Cols - 1
            'Set the value of each column in the row buffer
            'to the corresponding value in the arrray
            RowBuf.Value(i, j) = myarray(p, j)
         Next
         'set the value of the bookmark for
         'the current row in the rowbuffer
         RowBuf.Bookmark(i) = p

      Case 1   'Read bookmarks only
         'set the value of the bookmark for
```

```
            'the current row in the rowbuffer
            RowBuf.Bookmark(i) = p

         Case 2   'Read bookmarks and bound column
            'this just assumes the first column is the bound column
            RowBuf.Value(i, 0) = myarray(p, 0)
            'set the value of the bookmark for
            'the current row in the rowbuffer
            RowBuf.Bookmark(i) = p

         Case 3   'Read bookmarks and data for displayed column only
         End Select

         'move the pointer forward or backward, depending
         'on which way it's supposed to move
         If ReadPriorRows Then
            p = p - 1
         Else
            p = p + 1
         End If

         'increment the number of rows read
         r = r + 1
   Next i
```

ReBind Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

In bound mode, re-requests the data from the data source.

In unbound mode, refreshes the unbound control by rereading data from the top of the underlying data.

Syntax

object . **ReBind**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.

Remarks

In bound mode, invoking this method destroys column and group layout information before reconnecting to the data source.

In unbound mode, this method preserves column and group layout information, and simply reloads data by generating the **UnboundReadData** event.

This method does not apply to AddItem mode.

The **InitColumnProps** event is not generated when this method is invoked.

Example

The following code shows how to change the recordset an Unbound grid is displaying and then uses the Rebind Method to tell the grid to re-populate with the new data. Originally, the grid below is displaying the Titles table of the `BIBLIO.MDB` Database.

```
Dim i As Integer

'Change the recordset
Set db = OpenDatabase("e:\samples\biblio.mdb")
Set rs = db.OpenRecordset("Authors")

'ct is a count of the number of fields in the recordset. It is
'used in the UnboundReadData event to determine how many columns
'to populate in the grid. By setting it to 0, we prevent the grid from
loading data.
ct = 0

'Remove all the existing columns from the grid and add in the new
'columns to reflect the new data.
SSDBGrid1.Columns.RemoveAll

For i = 0 To (rs.Fields.Count - 1)
   SSDBGrid1.Columns.Add i
   SSDBGrid1.Columns(i).Caption = rs.Fields(i).Name
Next i

'Once the grid has all the columns necessary to display the new
'data, we must set ct to the number of fields in the recordset.
ct = rs.Fields.Count

'Rebind the grid. This causes the UnboundReadData to fire and re-
'populate the grid with data from the new recordset.
SSDBGrid1.ReBind
```

RecordSelectors Property

Applies To

SSDBGrid

Description

Determines whether record selectors will be displayed.

Syntax

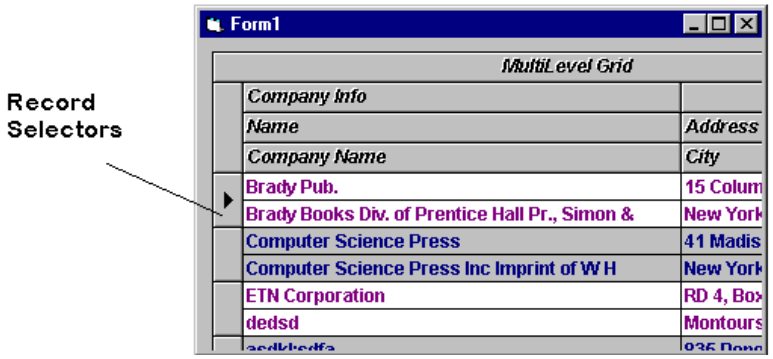
object . **RecordSelectors**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether record selectors are displayed, as described in Settings.

Settings

Setting	Description
True	(Default) Record selectors will be displayed.
False	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.

Redraw Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Returns or sets a value that determines when the control should repaint itself.

Syntax

object . **Redraw**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether the control should be redrawn, as described in Settings.

Settings

Setting	Description
True	(Default) The control repaints itself whenever necessary.
False	Repainting of the control is suspended until **Redraw** is set to True.

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 caching to improve performance during block additions. When **Redraw** is set to True, any cached data becomes available in the grid.

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

Applies To

Bookmarks collection, Buttons collection, Columns collection, Groups collection, SelBookmarks collection, StyleSets collection

Description

Used to remove a specified object from a collection.

Syntax

object . **Remove(***Index* **As Variant)**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
Index	Required. An expression that evaluates to a valid item to be removed from its collection.

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, the indices of all objects that come after the deleted object will be shifted up.

See Also

Add method, Count property, RemoveAll method

Example

The following example illustrates the use of the remove method by deleting the fourth button in a collection of ten buttons:

```
SSDBOptSet1.Buttons(4).Caption = "Fifth"
'Fourth button is removed
SSDBOptSet1.Buttons.Remove 3
'X contains "Fifth" since all buttons shifted
X = SSDBOptSet1.Buttons(3).Caption
```

RemoveAll Method (AddItem Mode)

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Removes all rows from an AddItem grid.

Syntax

object . **RemoveAll**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.

Remarks

If a row is currently being edited, it will not be removed. To resolve this issue, invoke the **CancelUpdate** method before this method.

Example

The following code removes all rows from an AddItem grid:

```
SSDBGrid1.RemoveAll
```

RemoveAll Method (Collections)

Applies To

Bookmarks collection, Buttons collection, Columns collection, Groups collection, SelBookmarks collection, StyleSets collection

Description

Removes all objects from a collection.

Syntax

object . **RemoveAll**

Part	Description
object	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.

See Also

Add method, Remove method

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)

Applies To

Column object

Description

Removes all items from a column's combo box.

Syntax

object . **RemoveAll**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.

Example

The following code example removes all items from a combo box in the third column:

```
SSDBGrid.Columns(2).RemoveAll
```

RemoveItem Method (AddItem Mode)

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Removes the row at a specified absolute number from an AddItem grid.

Syntax

object . **RemoveItem(** [*row* **As Variant**]**)**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
row	An integer specifying the absolute number of 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.

Do not attempt to use *row* in conjunction with the **Row** property, which refers to visible rows, and not absolute rows.

To delete rows that are selected, use the **DeleteSelected** method.

See Also

AddItemRowIndex method, DeleteSelected method

Example

The following code removes the current row of the grid.

```
SSDBGrid1.RemoveItem _
    SSDBGrid1.AddItemRowIndex(SSDBGrid1.Bookmark)
```

Note that the Row property of the grid is not used. This is because the RemoveItem method accepts an *absolute* row number, and the Row property provides only a *visible* row number.

RemoveItem Method (Column Object)

Applies To

Column object

Description

Removes an item from a combo box.

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
index	An integer specifying the index number of the item to remove.

Example

The following code example removes the fourth item of a combo box in the first column:

```
SSDBGrid1.Columns(0).RemoveItem(3)
```

Reset Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Destroys the associated layout for a control.

Syntax

object . **Reset**

Part	Description
object	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.Columns.Add 0
SSDBGrid1.Columns.Add 1
SSDBGrid1.Columns(0).Caption = "Name"
SSDBGrid1.Columns(1).Caption = "Social Security Number"
```

ResizeHeight Property

Applies To

SSDBGrid

Description

Sets or returns the height of rows after the user resizes a row.

Syntax

object . **ResizeHeight**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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 respond to or limit the sizing of rows.

See Also

RowResize, ResizeWidth

ResizeWidth Property

Applies To

SSDBGrid

Description

Sets or returns the width of a group or column after the user resizes it.

Syntax

object . **ResizeWidth**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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 respond to or limit the sizing of columns or groups.

See Also

ColResize, GrpResize, ResizeHeight

RotateText Property

Applies To

SSDBData

Description

Determines whether caption text should be rotated.

Syntax

object . **RotateText**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether caption area text should be rotated, as described in Settings.

Settings

Setting	Description
True	Caption area text will be rotated.
False	(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 See Also

Caption, Orientation

RoundedCorners Property

Applies To

SSDBCommand, SSDBData

Description

Determines whether the control should be displayed with rounded corners or square corners.

Syntax

object . **RoundedCorners**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether to display rounded corners, as described in Settings.

Settings

Setting	Description
False	Control will be displayed with square corners.
True	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:

Row Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Sets or returns the current visible row (not the absolute row number).

Syntax

object . **Row**[= *rownumber*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
rownumber	An integer expression specifying the current row (e.g., the first visble row is row 0).

Remarks

This property is only available at runtime.

The **Row** property reflects the value of the visible row, not the absolute row. See Upgrading to Data Widgets 3.1 for more information on the difference between visible and absolute row numbers.

To bring record into view, you have to set the current bookmark. The **Row** property will only affect a row currently being displayed.

This value is zero based.

See Also

Col, Grp, VisibleRows

RowAutoSize Property

Applies To

ssPrintInfo Object

Description

Determines whether the rows in a printed report will be resized automatically.

Syntax

object.**RowAutoSize**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying how row size will be adjusted, as described in Settings.

Settings

Setting	Description
True	Each row in the report will be sized to accomodate the data in that row. Long text fields will expand the height of the row so that all text appears.
False	(Default) All rows in the report will be the same size, as determined by the grid layout. Long text fields will be clipped.

Remarks

When **RowAutoSize** is set to False (the default) and you print a report, the size of the row is automatically determined by your grid layout and the size of the font used. If you have a long text field (such as a memo field) the report will print only the amount of text that can fit within the column on a single line. The rest of the text of that field will be cropped.

With **RowAutoSize** to True, the control will automatically adjust the height of each row to accomodate all the text in a long text field. The text field will be wrapped within the width of its column, and the row height of the record will be increased until all the text in the field is printed. Row height can only be extended to the bottom of the current page; after that, the text will be cropped.

You can control the maximum height of any given row using the **MaxLinesPerRow** property. This property gives you a way to limit the number of lines of text that will appear in any row.

See Also

MaxLinesPerRow

Example

The **PrintInitialize** event fires before the beginning of a print job to give you a way to set up the values that will be be displayed to the user in the Print and Print Setup dialog boxes. The **PrintBegin** event fires just before

printing actually begins (after any print dialogs have been displayed).

```
Private Sub SSDBGrid1_PrintInitialize(ByVal ssPrintInfo As ssPrintInfo)

   'Define a page header that includes a page number
   ssPrintInfo.PageHeader = "Print Sample: Page <page number>"
   'Define a page footer that specifies when the grid was printed.
   ssPrintInfo.PageFooter = "Printed <date> <time>"

   'Specify that we want each row's height
   'to expand so that all data is displayed,
   'but up to a maximum of 10 lines.
   ssPrintInfo.RowAutoSize = True      'So rows are expanded in height
                                       'as necessary
   ssPrintInfo.MaxLinesPerRow = 10     'but up to a maximum of 10
                                       'lines.

   'Indicate that we do not want to print colors.
   '(Use True to print colors or shades
   'of gray on a non-color printer).
   ssPrintInfo.PrintColors = False

   'Print the column headers.
   ssPrintInfo.PrintColumnHeaders = ssUseCaption

   'The only grid lines we want to print are to separate groups.
   '(There are other options to print various types of grid lines.)
   ssPrintInfo.PrintGridLines = ssPrintGridLinesColGrpHeadsOnly

   'Print column and group headers at the top of each page.
   '(Use ssTopOfReport if you want the headers
   'to appear on the first page.)
   ssPrintInfo.PrintHeaders = ssTopOfPage

End Sub
```

RowBookmark Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Returns a bookmark of a row in the grid's display area.

Syntax

object . **RowBookmark(***RowNumb* **As Long)**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
RowNumb	A long integer specifying the number of a visible row (e.g., the first visble row is row 0) of the bookmark to retrieve.

Remarks

Use this method to retrieve a bookmark to a row that is currently visible.

If you are unfamiliar with the concepts used when working with bookmarks, you might want to peruse the Bookmarks Tutorial.

See Also

RowContaining

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
```

RowChanged Property

Applies To

SSDBGrid

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*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether data in the row has changed, as described in Settings.

Settings

Setting	Description
True	Data in row has changed.
False	Data in row has not changed.

Remarks

When a new row is read into the grid, this property is automatically set to false.

RowColChange Event

Applies To

SSDBGrid

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
LastRow	A variant identifying the previous row before the change. If the row has not changed, this will equal the current row.
LastCol	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.

See Also

BeforeRowColChange, RowLoaded

RowContaining Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Returns the index of the visible row located at a specified y-coordinate.

Syntax

object . **RowContaining(***Y* **As Single)**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
Y	A floating point variable specifying the index of the row.

Remarks

The value range is from 0 to one less the value of **VisibleRows**.

If the specified coordinate is out of range, an error occurs. To ensure that the mouse pointer is over a row, use the **WhereIs** method.

See Also

ColContaining, Row, RowBookmark, WhereIs

Example

The following code changes the ToolTipText of the grid depending on which row the mouse pointer is over.

```
Private Sub SSDBGrid1_MouseMove(Button As Integer, Shift As Integer, x
As Single, y As Single)
    Dim VisibleRow As Integer
    Dim BookMark As Variant
    Dim ToolTip As String

    If SSDBGrid1.WhereIs(x, y) = ssWhereIsData Then
        'Get the Visible Row Number
        VisibleRow = SSDBGrid1.RowContaining(y)
        'Get the bookmark of the row
        BookMark = SSDBGrid1.RowBookmark(VisibleRow)
        'Set the tooltip of the Grid to the
        'first column of the row the
        'mouse is currently over.
        ToolTip = SSDBGrid1.Columns(0).CellText(Bookmark)
    End If
    SSDBGrid1.ToolTipText = ToolTip
End Sub
```

RowCount Property

Applies To

ssRowBuffer object

Description

Returns the total number of rows requested by the ssRowBuffer object.

Syntax

object . **RowCount** [= *integer*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
integer	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 in the **UnboundReadData** event is fixed at a maximum of 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 **RowCount** of the **RowBuffer** object is set to less than its previous value. This signals the control that all of the data has been read, and it will not fire another **UnboundReadData** event.

See Also

UnboundReadData event

RowExport Event

Applies To

SSDBGrid

Description

Occurs just before a grid row is exported.

Syntax

Sub *control*_**RowExport(***Bookmark* **As Variant**, *Cancel* **As Integer)**

The event parameters are:

Parameter	Description
Bookmark	A variant expression that evaluates to the bookmark of the row that is about to be printed.
Cancel	A Boolean expression that determines whether the export will continue, as described in Settings.

Settings

Setting	Description
True	The export is stopped. The current row will not be exported, nor will any rows that follow.
False	(Default) The row will be included in the output file and the export of data will continue.

Remarks

A single- or multiple-row export is initiated by invoking the **Export** method. The **RowExport** event occurs multiple times during a multiple row export - once for each row included in the output file - or once for a single row export. It is similar to the **RowLoaded** event, which also occurs multiple times during export, just before the **RowExport** event.

RowExport gives you the ability to examine the data that is about to be exported and to change it if necessary. It also gives you the ability to stop the export by setting the value of the *Cancel* parameter to True. You can use this to examine exported data a row at a time and stop the export when certain criteria are met, such as the occurrence of a certain record. Any rows exported prior to the cancelled **RowExport** event are successfully written to the output file.

Note Changes made to data during the **RowExport** event, such as `SSDBGrid1.Columns(1).Text = "New Text"`, appear only in the exported data. They are not reflected in the displayed grid.

See Also

Export

Example

This example shows how you can examine the contents of a row that is about to be exported and optionally halt the export based on the information:

```
Private Sub SSDBGrid1_RowExport(ByVal Bookmark As Variant, Cancel As
Integer)

   'This example stops exporting rows if a cell
   'in column 1 is less than 0.
   If Val(SSDBGrid1.Columns(1).Text) < 0 Then
      Cancel = True
      MsgBox "An invalid value is in the grid!", vbOKOnly, "Alert!"
   End If

End Sub
```

RowHeight Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Sets or returns the height of the rows.

Syntax

object . **RowHeight**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the height of every row.

Remarks

All rows are the same height.

See Also

DefColWidth

RowLoaded Event

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Occurs when the control is about to display a row of data in the visible area.

Syntax

Sub control_**RowLoaded** (**ByVal***Bookmark* **As Variant**)

The event parameters are:

Parameter	Description
Bookmark	A variant specifying the bookmark of the row that has been loaded.

Remarks

Most commonly, this event is used to populate an unbound column in a bound control, retrieve the total value of a column's contents in AddItem mode, and apply **StyleSets** based on a particular condition.

This event is also generated when a row is updated.

Certain properties operate differently inside the **RowLoaded** event than outside of it. In general, properties that affect the current row under normal circumstances will operate on the row being loaded when used within the **RowLoaded** event procedure.Such is the case with the **Text** and **Value** properties, as well as the **CellStyleSet** method Properties or methods that behave differently within the **RowLoaded** event procedure have their distinguished use noted in the Remarks section of their description.

The row being loaded (and the one on which properties will operate) is indicated by the *Bookmark* parameter passed to the event.

See Also

CellStyleSet, RowColChange, Text, Value

Example

In the following code demonstrates populating an Unbound column in a bound DataGrid. The column is populated with the sum of two other columns in the same row. In this code, Column 2 is the Unbound column. Columns 0 and 1 are bound and get their data from a recordset.

In the SSDBgrid1_RowLoaded event:

```
SSDBGrid1.Columns(2).Value = SSDBGrid1.Columns(0).Value_
   + SSDBGrid1.Columns(1).Value
```

Example (StyleSets)

In the following code, different stylesets are applied to a row in the grid, depending on the value in Column 0. If the Value is negative, a Styleset named "Red" is applied to the row. If the value is positive a Styleset called "Black" is applied.

In the SSDBgrid1_InitColumnProps event:

```
SSDBGrid1.StyleSets.Add "Red"
SSDBGrid1.StyleSet("Red").ForeColor = vbRed
SSDBGrid1.StyleSets.Add "Black"
SSDBGrid1.StyleSet("Black").ForeColor = vbBlack
```

In the SSDBgrid1_RowLoaded event:

```
Dim i As Integer
Dim StyleSetName As String

Select Case SSDBGrid1.Columns(0).Value
Case Is < 0
   StyleSetName = "Red"
Case Else
   StyleSetName = "Black"
End Select

For i = 0 To SSDBGrid1.Cols - 1
   SSDBGrid1.Columns(i).CellStyleSet StyleSetName
Next i
```

RowNavigation Property

Applies To

SSDBGrid

Description

Determines how the arrow keys respond when navigating rows in the grid.

Syntax

object . **RowNavigation**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

See Also

CellNavigation

RowOffset Property

Applies To

Button object, SSDBOptSet

Description

Determines the vertical offset used to draw the button.

Syntax

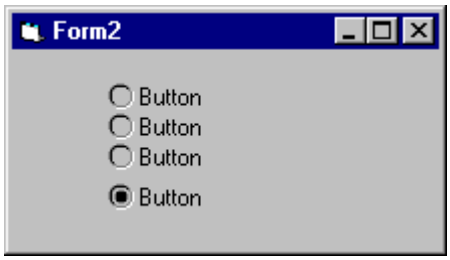
object . **RowOffset**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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:



See Also

ColOffSet

RowPrint Event

Applies To

SSDBGrid

Description

Occurs just before a grid row is printed.

Syntax

Sub *control*_**RowPrint(***Bookmark* **As Variant**, *PageNumber* **As Long**, *Cancel* **As Integer)**

The event parameters are:

Parameter	Description
Bookmark	A variant expression that evaluates to the bookmark of the row that is about to be printed.
PageNumber	A long integer expression that evaluates to the number of the page containing the row about to be printed.
Cancel	A Boolean expression that determines whether printing will continue, as described in Settings.

Settings

Setting	Description
True	Printing will be stopped. The current row will not be printed, nor will any rows that follow it.
False	(Default) The row will be printed and printing will continue.

Remarks

Printing is initiated by invoking the **PrintData** method. The **RowPrint** event occurs multiple times during the printing of a report - once for each row included in the report. It is similar to the **RowLoaded** event, which also occurs multiple times during printing, just before the **RowPrint** event.

RowPrint gives you the ability to examine the data that is about to be printed and to change it if necessary. It also gives you information about which page a row will be printed on via the *PageNumber* parameter, and allows you to cancel the printing of the report by setting the value of the *Cancel* parameter to True. You can use this to examine exported data a row at a time and stop the report when certain criteria are met, such as the occurrence of a certain record. Any rows printed prior to the cancelled **RowPrint** event are included in the report.

Note Changes made to data during the **RowPrint** event, such as `SSDBGrid1.Columns(1).Text = "New Text"`, appear only in the printed report. They are not reflected in the displayed grid.

See Also

PrintBegin, PrintData, PrintError, PrintInitialize, ssPrintInfo

Example

This code uses the RowPrint event to test for the presence of a value, and modifies the printed data based on that value.

```
Private Sub SSDBGrid1_RowPrint(ByVal Bookmark As Variant, ByVal PageNumber
As Long, Cancel As Integer)

   'This example checks for "Secret" in the first column
   'and replaces any data in that row with asterisks in the
   'printed report. It does not affect the visible grid.

   If SSDBGrid1.Columns(0).Text = "Secret" Then
      For iC = 0 to (SSDBGrid1.Columns.Count - 1)
         iCellLen = Len(SSDBGrid1.Columns(iC).Text)
         SSDBGrid1.Columns(iC).Text = String(iCellLen, "*")
      Next iC
   End If

End Sub
```

RowResize Event

Applies To

SSDBGrid

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.

Rows Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Sets or returns the number of rows in the grid.

Syntax

object . **Rows**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the number of rows.

Remarks

At runtime, **Rows** is read-only in Bound and AddItem modes. In Unbound mode, at runtime, however, the control uses the value of this property to automatically adjust its vertical scrollbar.

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.

See Also

Col, Cols, UseExactRowCounnt, Row, UnboundReadData

Example

In the following code, the Rows property of an Unbound grid is set in order to adjust the grid's scrollbars.

In the Form_Load event:

```
Dim i As Integer

Set db = OpenDatabase(App.Path & "\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

SSDBGrid1.Rows = rs.RecordCount
```

RowSelectionStyle Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Determines how a row will appear when selected.

Syntax

object . **RowSelectionStyle**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

See Also

ActiveRowStyleSet

RowTop Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Returns the y-coordinate of the top of a row.

Syntax

object . **RowTop(***RowNum* **As Integer)**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
rownum	An integer specifying the row number.

SavedBookmark Property

Applies To

SSDBCommand

Description

Sets or returns the currently saved bookmark.

Syntax

object . **SavedBookmark**[= *value*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
Value	A variant specifying the bookmark value.

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.

SaveLayout Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Saves the indicated portion of the control's layout to the specified file.

Syntax

object.**SaveLayout(***FileName* **As String**, *Flags* **As Integer)**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
FileName	A string expression that evaluates to the file name of the layout file to create. Layout files end with the extension ".GRD".
Flags	An enumerated integer expression specifying what part of the layout to save, as described in Settings.

Settings

Setting	Description
1	*ssSaveLayoutStylesetsOnly*. Only the StyleSets from the control will be saved in the layout file.
2	*ssSaveLayoutAll*. All formatting attributes from the control will be saved.

Remarks

The **SaveLayout** method saves the current grid layout to the file specified by the *FileName* argument. Depending on the setting of *Flags*, either the entire layout or only the StyleSets will be saved.

Grid layouts of either type can be saved from the Grid Editor by clicking the "Save..." button, specifying which type of file to create and supplying a file name.

Grid layouts are saved as files. Layout files can have any name; the default extension for a grid layout file is .GRD.

Note Fonts and pictures saved in a layout file by a 32-bit program cannot be read by a 16-bit program. If you create a .GRD file using the 32-bit version of DataWidgets, then load the file using the 16-bit version, font and picture information will not be applied.

The following is a summary of which attributes are saved when using the **SaveLayout** method. Any attributes saved will be overwritten in the grid when the **LoadLayout** method is invoked with the filename of a valid grid layout file.

StyleSets (saved when flags = *ssSaveLayoutStylesetsOnly* or *ssSaveLayoutAll*)

AlignmentPicture	**Font**	**Picture**
AlignmentText	**ForeColor**	**PictureMetaHeight**
BackColor	**Name**	**PictureMetaWidth**

Shared Control Level Properties (saved when flags = *ssSaveLayoutAll*)

BackColor	DividerStyle	HeadFont
BackColorEven	DividerType	HeadFont3D
BackColorOdd	Enabled	HeadLines
BevelColorFace	FieldDelimiter	LevelCount
BevelColorFrame	FieldSeparator	RowHeight
BevelColorHighlight	Font	RowSelectionStyle
BevelColorScheme	Font3D	ScrollBars
BevelColorShadow	ForeColor	SplitterPos
BorderStyle	ForeColorEven	StyleSet
CheckBox3D	ForeColorOdd	UseExactRowCount
ColumnHeaders	GroupHeaders	Visible
DefColWidth	GroupHeadLines	

Grid Control Level Properties (saved when flags = *ssSaveLayoutAll*)

AllowAddNew	AllowRowSizing	PictureComboButton
AllowColumnMoving	AllowUpdate	PictureRecordSelectors
AllowColumnShrinking	BalloonHelp	RecordSelectors
AllowColumnSizing	Caption	RowNavigation
AllowColumnSwapping	CaptionAlignment	SelectByCell
AllowDelete	CellNavigation	SelectTypeCol
AllowDragDrop	MaxSelectedRows	SelectTypeRow
AllowGroupMoving	MultiLine	SplitterVisible
AllowGroupShrinking	PageFooterFont	TabNavigation
AllowGroupSizing	PageHeaderFont	UseDefaults
AllowGroupSwapping	PictureButton	

Combo Control Level Properties (saved when flags = *ssSaveLayoutAll*)

BevelType	GetListWidthAutoSize	PromptChar
BevelWidth	GetMultiLine	PromptInclude
ClipMode	ListWidth	Text
GetAllowInput	Mask	TextFormat
GetAllowNull	MaxDropDownItems	
GetAutoRestore	MinDropDownItems	
GetListAutoPosition	MultiLine	
GetListAutoValidate	PictureDropDown	

DropDown Control Level Properties (saved when flags = *ssSaveLayoutAll*)

DataFieldList	ListWidth	MinDropDownItems
DataFieldToDisplay	MaxDropDownItems	

Groups (saved when flags = *ssSaveLayoutAll*)

AllowSizing	HasHeadForeColor	StyleSet
Caption	HeadBackColor	Visible
CaptionAlignment	HeadForeColor	Width
HasHeadBackColor	HeadStyleSet	

Columns (saved when flags = *ssSaveLayoutAll*)

Alignment	HasBackColor	Locked
AllowSizing	HasForeColor	Name
BackColor	HasHeadBackColor	NumberFormat
ButtonsAlways	HasHeadForeColor	Position
Caption	HeadBackColor	Style
CaptionAlignment	HeadForeColor	StyleSet
Case	HeadStyleSet	VertScrollBar
FieldLen	Level	Visible
ForeColor	List	Width
Group	ListCount	

List Items (saved when flags = *ssSaveLayoutAll* only for columns with **Style** property set to *ssStyleComboBox*)

Item	ItemData

See Also

LoadLayout

Example

This routine saves the current layout, loads a layout used for printing,then restores the original layout again. You may want to do this to use a different color scheme for printing than the one shown on the screen.

```
Private Sub PrintGridUsingPrintLayout()

   On Error GoTo layout_err_1
```

```
    'Save current grid layout.
    SSDBGrid1.SaveLayout App.Path + "\User Layout.grd", ssSaveLayoutAll
    'Load the print layout.
    SSDBGrid1.LoadLayout App.Path + "\Print Layout.grd"
    'Print the grid.
    SSDBGrid1.PrintData ssPrintAllRows, False, True
    'Restore the user's layout
    SSDBGrid1.LoadLayout App.Path + "\User Layout.grd"

Exit Sub

layout_err_1:
   MsgBox Err.Description

End Sub
```

Scroll Event

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Occurs just before a scroll takes place.

Syntax

Sub control_**Scroll** (*Cancel* **As Integer**)

The event parameters are:

Parameter	Description
Cancel	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 occuring by setting the Cancel parameter to **True**. If this event is not cancelled, the scroll occurs and the **ScrollAfter** event is fired.

See Also

Scrollbars, Scroll method, ScrollAfter event

Scroll Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Causes the grid to be scrolled.

Syntax

object . **Scroll** *Cols* **As Integer**, *Rows* **As Long**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
Cols	An integer expression specifying how many columns to scroll.
Rows	A long integer expression specifying how many rows to scroll.

See Also

Scrollbars, Scroll event

ScrollAfter Event

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

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 occuring by setting the Cancel parameter of the **Scroll** event to **True**.

See Also

Scrollbars, Scroll method, ScrollAfter event

Scrollbars Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Determines the type of scrollbars to use.

Syntax

object . **Scrollbars**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

See Also

Scroll event, Scroll method

SelBookmarks Collection

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

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, highlighting that row. Depending on the value of the control's **SelectTypeRow** property, other rows may be removed from the collection when adding a row to the collection.

If the value of the **SelectTypeRow** property permits the selction of multiple rows, each row selected will be added to the collection in the order in which the selection occurred; order is not based on the displayed order. When a row is deselected, that row is removed from the **SelBookmarks** collection.

You can add bookmarks to, and remove them from the **SelBookmarks** collection programmatically. Rows may also be easily accessed in the **SelBookmarks** collection without moving the current row position.

See Also

MaxSelectedRows property, RowSelectionStyle property

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 - 1

' 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")
```

SelBookmarks Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Returns a Bookmark object at the specified index.

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
Index	A variant specifying the bookmark number.

Remarks

When no index is specified the SelBookmarks collection object is returned.

See Also

DeleteSelected

SelChange Event

Applies To

SSDBGrid

Description

Occurs when the current selection 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:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
seltype	Indicates the type of selection (0=Group, 1=Column, 2=Row).
cancel	Determines whether the selection reverts to its position before the event occurred.
DispSelRowOverflow	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 whenever the selection of a range of cells changes, for example is a row is selected and the user clicks on another row, or if a column is selected and the user clicks on a cell in another column. If there is no selection, the event will not be fired.

The firing of this event will be affected by the types of selection allowed in the control, according to the settings of **SelectByCell**, **SelectTypeCol** and **SelectTypeRow**.

Setting *cancel* to True causes the selection to revert to the cell or range active before the event occurred.

See Also

SelectByCell property, SelectTypeCol property, SelectTypeRow property

SelectByCell Property

Applies To

SSDBGrid

Description

Determines if selection of the row will occur if the user clicks on a cell.

Syntax

object . **SelectByCell**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying if the entire row should be selected when a user clicks on a cell, as described in Settings.

Settings

Setting	Description
True	The row will be selected when a user clicks on a cell in the row.

False	(Default) 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.

See Also

SelectTypeCol, SelectTypeRow

Selected Property

Applies To

Column object, Group object

Description

Sets or returns whether an object is selected.

Syntax

object . **Selected**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether the object is selected, as described in Settings.

Settings

Setting	Description
True	Object is currently selected.
False	Object is not currently selected.

SelectTypeCol Property

Applies To

SSDBGrid

Description

Determines the column selection type.

Syntax

object . **SelectTypeCol**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

See Also

SelectTypeRow

SelectTypeRow Property

Applies To

SSDBGrid

Description

Determines the row selection type.

Syntax

object . **SelectTypeRow**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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

For **SelectTypeRow** 0 (None), 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.

For **SelectTypeRow** 3 (Multi-Select), multiple selections can be made using the SHIFT and CTRL keys. SHIFT is used for selection of contiguous rows while CTRL is used for selection (and de-selection) of individual non-adjacent rows.

See Also

SelBookmarks collection, SelChange event, SelectTypeCol propery, MaxSelectedRows property

ShowAddButton Property

Applies To

SSDBData

Description

Determines whether the Add Record button is displayed on the control.

Syntax

object . **ShowAddButton**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying the display state of the Add Record button on the control, as described in Settings.

Settings

Setting	Description
True	(Default) The Add Record button will be displayed.
False	The Add Record button will not be displayed.

See Also

ShowBookmarkButtons, ShowCancelButton, ShowDeleteButton, ShowFindButtons, ShowFirstLastButtons, ShowPageButtons, ShowPrevNextButtons, ShowUpdateButton

ShowBookmarkButtons Property

Applies To

SSDBData

Description

Determines which of the bookmark buttons are to be displayed on the control.

Syntax

object . **ShowBookmarkButtons**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	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.

See Also

ShowAddButton, ShowCancelButton, ShowDeleteButton, ShowFindButtons, ShowFirstLastButtons, ShowPageButtons, ShowPrevNextButtons, ShowUpdateButton

ShowBookmarkDropDown Event

Applies To

SSDBData

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.

See Also

DroppedDown, CloseBookmarkDropDown Method

ShowCancelButton Property

Applies To

SSDBData

Description

Determines whether the Cancel button is displayed on the control.

Syntax

object . **ShowCancelButton**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying the display state of the Cancel button on the control.

Settings

Setting	Description
True	(Default) The Cancel button will be displayed.
False	The Cancel button will not be displayed.

Remarks

The Cancel (Cancel Add) button cancels the adding of a new record to the database.

See Also

ShowAddButton, ShowBookmarkButtons, ShowDeleteButton, ShowFindButtons, ShowFirstLastButtons, ShowPageButtons, ShowPrevNextButtons, ShowUpdateButton

ShowDeleteButton Property

Applies To

SSDBData

Description

Determines whether the Delete Record button is displayed on the control.

Syntax

object . **ShowDeleteButton**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying the display state of the Delete Record button on the control, as described in Settings.

Settings

Setting	Description
True	(Default) The Delete Record button will be displayed.
False	The Delete Record button will not be displayed.

See Also

ShowAddButton, ShowBookmarkButtons, ShowCancelButton, ShowFindButtons, ShowFirstLastButtons, ShowPageButtons, ShowPrevNextButtons, ShowUpdateButton

ShowFindButtons Property

Applies To

SSDBData

Description

Determines which of the find buttons are to be displayed on the control.

Syntax

object . **ShowFindButtons**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying which of the find buttons are to be displayed, as described in Settings.

Settings

Setting	Description
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.

See Also

ShowAddButton, ShowBookmarkButtons, ShowCancelButton, ShowDeleteButton, ShowFirstLastButtons, ShowPageButtons, ShowPrevNextButtons, ShowUpdateButton

ShowFindDialog Event

Applies To

SSDBData

Description

Occurs when the Find dialog is called, immediately prior to displaying the dialog.

Syntax

Sub control_**ShowFindDialog**()

See Also

FindDialog, CloseFindDialog event

ShowFirstLastButtons Property

Applies To

SSDBData

Description

Determines whether the first and last record button is displayed on the control.

Syntax

object . **ShowFirstLastButtons**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying the display state of the First Record and Last Record buttons on the control, as described in Settings.

Settings

Setting	Description
True	(Default) The First Record and Last Record buttons will be displayed.
False	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.

See Also

ShowAddButton, ShowBookmarkButtons, ShowCancelButton, ShowDeleteButton, ShowFindButtons, ShowPageButtons, ShowPrevNextButtons, ShowUpdateButton

ShowPageButtons Property

Applies To

SSDBData

Description

Determines whether the page forward and page backward buttons are displayed on the control.

Syntax

object . **ShowPageButtons**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.

boolean	A Boolean expression specifying the display state of the Page forward and Page backward buttons on the control, as described in Settings.

Settings

Setting	Description
True	(Default) The Page forward and Page backward buttons will be displayed.
False	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.

See Also

ShowAddButton, ShowBookmarkButtons, ShowCancelButton, ShowDeleteButton, ShowFindButtons, ShowFirstLastButtons, ShowPrevNextButtons, ShowUpdateButton

ShowPrevNextButtons Property

Applies To

SSDBData

Description

Determines whether the previous record and next record buttons are displayed on the control.

Syntax

object . **ShowPrevNextButtons**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying the display state of the Previous Record and Next Record buttons on the control, as described in Settings.

Settings

Setting	Description
True	(Default) The Previous Record and Next Record buttons will be displayed.
False	The Previous Record and Next Record buttons will not be displayed.

ShowPrevNextButtons Property See Also

ShowAddButton, ShowBookmarkButtons, ShowCancelButton, ShowDeleteButton, ShowFindButtons, ShowFirstLastButtons, ShowPageButtons, ShowUpdateButton

ShowUpdateButton Property

Applies To

SSDBData

Description

Determines whether the Update Record button is displayed on the control.

Syntax

object . **ShowUpdateButton**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying the display state of the Update Record button on the control, as described in Settings.

Settings

Setting	Description
True	(Default) The Update Record will be displayed.
False	The Update Record will not be displayed.

Remarks

The Update Record button will write any changes made to the most recently modified record.

See Also

ShowAddButton, ShowBookmarkButtons, ShowCancelButton, ShowDeleteButton, ShowFindButtons, ShowFirstLastButtons, ShowPageButtons, ShowPrevNextButtons

Size Property

Applies To

Font object, HeadFont object

Description

Returns or sets the font size used in the specified **Font** or **Headfont** object.

Syntax

object . **Size**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
Font	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.

Example

The following sample code sets the font size to 18 points:

```
SSDBOptSet.Font.Size = 18
SSDBGrid.Headfont.Size = 18
```

Soundex Method

Applies To

SSDBData

Description

Returns the soundex string for a supplied string.

Syntax

object . **Soundex(***String* **As String)**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
String	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 similiar.

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.

SplitterMove Event

Applies To

SSDBGrid

Description

Occurs when the splitter is relocated by the user.

Syntax

Sub control_**SplitterMove** (*Cancel* **As Integer**)

The event parameters are:

Parameter	Description
Cancel	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.

See Also

SplitterPos, SplitterVisible

SplitterPos Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Sets or returns the column in which to place the splitter.

Syntax

object . **SplitterPos**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the position of the splitter.

Remarks

Valid range is from 0 to the maximum number of columns created. Note that the position of the splitter is relative to the number of visible columns. Hidden columns have no effect on the value of **SplitterPos**.

See Also

SplitterVisible, SplitterMove Event

SplitterVisible Property

Applies To

SSDBGrid

Description

Determines whether the splitter is visible.

Syntax

object . **SplitterVisible**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying the display state of the splitter, as described in Settings.

Settings

Setting	Description
True	The splitter will be displayed.
False	(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.

See Also

SplitterPos, SplitterMove Event

ssPrintInfo Object

Description

The ssPrintInfo object contains information about a pending print job.

Properties

ClippingOverride	MaxLinesPerRow	PrintColumnHeaders
Collate	PageBreakOnGroups	PrinterDeviceName
Copies	PageEnd	PrinterDriverVer
DriverOverride	PageFooter	PrintGridLines
MarginBottom	PageHeader	PrintGroupHeaders
MarginLeft	PageStart	PrintHeaders
MarginRight	Portrait	RowAutoSize
MarginTop	PrintColors	

Remarks

The **ssPrintInfo** object contains information about a pending print job. The properties of the object may be set through code, or by the end user via the Print and Print Setup dialog boxes.

The ssPrintIno object is created by invoking the **PrintData** method. It is available only within the scope of two events: **PrintInitialize** and **PrintBegin**. The ssPrintInfo object is passed as a parameter to both of these events.

In the **PrintInitialize** event, you can set the properties of the ssPrintInfo object to determine the default settings for the print job. If you have specified in the **PrintData** method that the Print Setup or Print dialog should be displayed to the user, the settings applied during this event will determine the default values in the dialogs.

In the **PrintBegin** event, you can examine the properties of the ssPrintInfo object to see what changes the user has made via the Print Setup and Print dialogs, optionally storing the values for later use. You can also change the

properties of ssPrintInfo object if the user's selections are somehow unsuitable.

Note The DataGrid will *not* print graphics that are included as part of a styleset. Pictures in grid cells or captions will not appear when the data is printed.

See Also

PrintBegin event, PrintData method, PrintError event, PrintInitialize event, RowPrint event

ssRowBuffer Object

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Used to transfer data to and from the control in unbound mode.

Properties

Bookmark	ColumnName	RowCount
ColumnCount	ReadType	Value

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 of data required by the object. When **ReadType** equals either 2 or 3, **ColumnCount** will equal 1, since the only column required is the one specified by either **DataFieldList** or **DataFieldToDisplay**.

The **ColumnName** property returns the caption 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.

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.

See Also

ReadType property, UnboundAddData event, UnboundDeleteRow event, UnboundReadData Event

UnboundWriteData event, Performance Tuning

Strikethrough Property

Applies To

Font object, HeadFont Object

Description

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

Syntax

object . **Strikethrough**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
Font	A Boolean expression specifying the font style as described in Settings.

Settings

Setting	Description
True	Turns on strikethrough formatting
False	(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.

Example

The following sample code causes the text to appear with strikethrough formatting:

```
SSDBOptSet1.Font.Strikethrough = True
SSDBGrid.Headfont.Strikethrough = True
```

String Property (Bookmark Object)

Applies To

Bookmark object (SSDBData only)

Description

Returns the string stored in a bookmark object.

Syntax

object . **String**[= *string*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
text	A string expression that evaluates to the string stored in a bookmark.

See Also

Value

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"
```

Style Property

Applies To

Column object

Description

Sets or returns the column's control style.

Syntax

object . **Style**[= *number*]

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

Settings

Setting	Description
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.

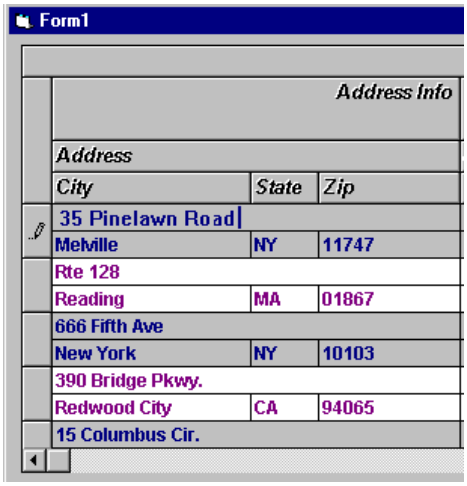
Remarks

When **Style** is 1 (Edit Button) or 4 (Button), the **BtnClick** event is generated (provided the **AllowUpdate** property is **True**).

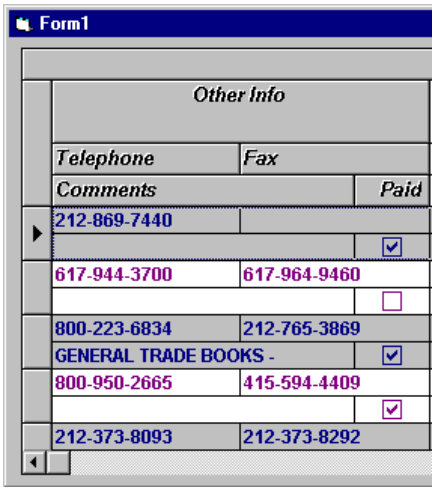
If **Style** is 2 (Check Box) and **AllowUpdate** is **True**, the **Change** event is generated when the box is checked (a value of **True** or -1) or unchecked (**False** or 0).

When **Style** is 3 (Combo Box), the cells behave similarly to the intinsic ComboBox controls, and provide additional properties, methods, and events that can be accessed via the **Column** object. These include the **List**, **ListIndex**, and **ItemData** properties, the **AddItem** and **RemoveItem** methods, and the **ComboDropDown** and **ComboCloseUp** events.

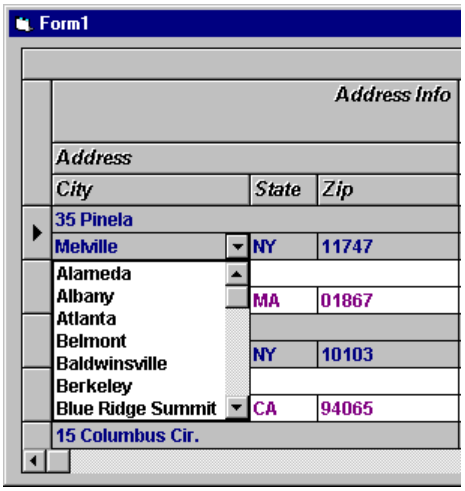
The following are examples of the various modes that **Style** can be set to:



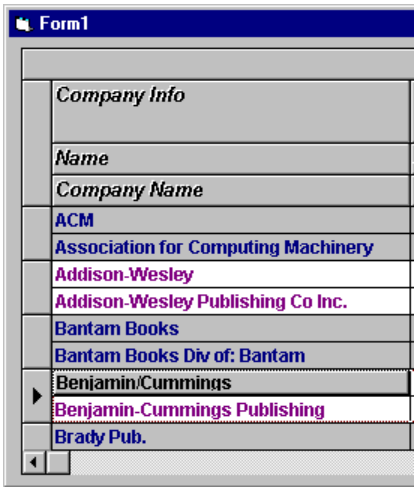
Edit (**Address** field)



CheckBox (**Paid** field)



Combo Box (**City** field)



Button (**Name** field)

Example

The following code makes Column 0 of a grid a ComboBox and then populates the ComboBox with 10 items.

```
Dim i As Integer

SSDBGrid1.Columns(0).Style = ssStyleComboBox
For i = 1 To 10
   SSDBGrid1.Columns(0).AddItem "Item #" & i
Next i
```

StyleSet Object

Applies To

StyleSets collection

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

AlignmentPicture	ForeColor	PictureMetaHeight
AlignmentText	Name	PictureMetaWidth
BackColor	Picture	
Font		

Remarks

You identify a **StyleSet** by using the **StyleSet** or **HeadStyleSet** properties.

Note that referencing a non-existent **StyleSet** or its properties will automatically create it.

See Also

ActiveRowStyleSet, CellStyleSet, HeadStyleSet, StyleSet, StyleSets collection.

StyleSet Property

Applies To

ActiveCell object, Column object, Group object, SSDBCombo, SSDBDropDown, SSDBGrid

Description

Returns or sets the name of a **StyleSet** in the **StyleSets** collection.

Syntax

object . **StyleSet**[= *text*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
text	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 by invoking the control's **Refresh** method. StyleSets will override each other based on the following hierarchy:

Data Area:

ActiveCell.StyleSet	(overrides all below)
Control. ActiveRowStyleSet	(overrides all below)
Column. StyleSet	(overrides all below)
Group.StyleSet	(overrides all below)
Control.StyleSet	

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

See Also

ActiveRowStyleSet, CellStyleSet, HeadStyleSet, StyleSet object, StyleSets collection

StyleSets Collection

Applies To

Column object, Group object, SSDBCombo, SSDBDropDown, SSDBGrid

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.

See Also

HeadStyleSet, StyleSet, StyleSet object

StyleSets Method

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Returns a StyleSet object at the specified index.

Syntax

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

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
Index	A variant specifying the StyleSet number.

Remarks

When no index is specified the StyleSets collection object is returned.

TabNavigation Property

Applies To

SSDBGrid

Description

Determines how the grid will process the Tab key.

Syntax

object . **TabNavigation**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression that specifies how the control will handle Tab key usage, as described in Settings.

Settings

Setting	Description
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.

TagVariant Property

Applies To

Column object, Group object, SSDBCombo, SSDBCommand, SSDBDropDown, SSDBGrid, SSDBOptSet

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*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
text	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.

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
```

Text Property

Applies To

ActiveCell object, Column object, SSDBCombo

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*]

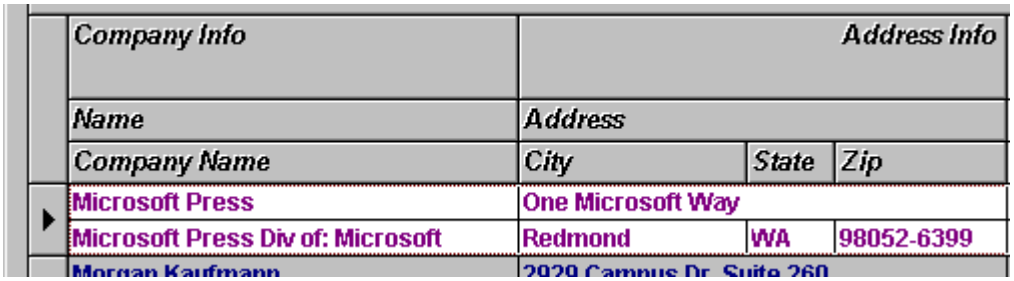
Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
text	A string expression that evaluates to the string stored in the column.

Remarks

As opposed to the **CellText** method, this property returns the value currently in place, and not the value from the underlying data.

Text returns the value as a string, while **Value** returns the value in its native data format.

The **DataFieldList** and **DataFieldToDisplay** property settings of the SSDBCombo affect the value returned by the **Text** property.



	Company Info	Address Info		
	Name	Address		
	Company Name	City	State	Zip
▶	Microsoft Press	One Microsoft Way		
	Microsoft Press Div of: Microsoft	Redmond	WA	98052-6399
	Morgan Kaufmann	2929 Campus Dr. Suite 260		

Based on the above graphic, the following is true:

```
SSDBGrid.Columns(0).Text = "Microsoft Press"
SSDBGrid.Columns(3).Text = "Redmond"
```

See Also

CellText method, CellValue method, Value property

Example

The following code shows how to programmatically "choose" the first item from a DataCombo.

```
SSDBCombo1.MoveFirst
SSDBCombo1.Text = SSDBCombo1.Columns(1).Text
```

TextError Event

Applies To

SSDBCombo, SSDBDropDown

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
ErrCode	A long integer specifying the error code resulting from the validation failure.
ErrString	A string that evaluates to the error message that SSDBCombo generates.
RestoreString	A string that evaluates to the restore string if RtnRestore is true.
Text	A string that evaluates to the text that failed validation.
RtnDispErrMsg	An integer expression that indicates if an error message box should be displayed.
RtnRestore	An integer expression that determines if text should be restored to its original state.

See Also

ListAutoValidate

TextFormat Property

Applies To

SSDBCombo

Description

Used to set the format mask of the edit portion of the control.

Syntax

object . **TextFormat**[= *string*]

Settings

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
string	A string expression that determines the format mask to use.

Remarks

This property complies with standard text formatting properties.

Example

In this example, data displayed in the edit portion of the SSDBCombo will be displayed as currency:

```
SSDBCombo1.TextFormat = "Currency"
```

UnboundAddData Event

Applies To

SSDBGrid

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 contains the new row's 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.

The **UboundReadData** event will generated immediately following this event in order to update the grid's display. Therefore, ensure that the underlying data used within the **UnboundReadData** event procedure is updated.

See Also

AllowAddNew property, UnboundPositionData event, UnboundReadData event, UnboundWriteData event, ssRowBuffer Object

Example

Exercise 2 and Exercise 3 of the Data Grid Guided Tours contain complete code examples of how to use this event. Examples include using Unbound mode to connect the Grid to a Recordset object and using Unbound mode to connect the Grid to a memory array.

UnboundDeleteRow Event

Applies To

SSDBGrid

Description

Occurs when a row is deleted from an unbound grid.

Syntax

Sub control_**UnboundDeleteRow** (*Bookmark* **As Variant**)

The event parameters are:

Parameter	Description
Bookmark	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 is automatically set to the bookmark value provided when the row was retrieved with the **UnboundReadData** event or added by **UnboundAddData** event.

See Also

AllowDelete property, UnboundAddData event, UnboundPositionData event, UnboundReadData event, UnboundWriteData event, ssRowBuffer Object

Example

Exercise 2 and Exercise 3 of the Data Grid Guided Tours contain complete code examples of how to use this event. Examples include using Unbound mode to connect the Grid to a Recordset object and using Unbound mode to connect the Grid to a memory array.

UnboundPositionData Event

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Occurs when a user repositions an unbound control.

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 that was current before the repositioning took place.
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 (such as when using the scroll bar's thumb to scroll). *StartLocation* and

NumberOfRowsToMove are provided by the control so that *NewLocation* can be specified.

The **Rows** property should be set to the total number of records in the underlying data.

See Also

UnboundAddData event, UnboundDeleteRow event, UnboundReadData event, UnboundWriteData event, ssRowBuffer Object

Example

Exercise 2 and Exercise 3 of the Data Grid Guided Tours contain complete code examples of how to use this event. Examples include using Unbound mode to connect the Grid to a Recordset object and using Unbound mode to connect the Grid to a memory array.

UnboundReadData Event

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

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
RowBuf	The ssRowBuffer object that will contain the retrieved data. The number of rows to be retrieved is determined by **RowCount**.
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.
ReadPriorRows	If set to **False**, data is retrieved in a forward direction, if **True**, data is retrieved in a reverse direction.

Remarks

This event notifies you that the control must retrieve data. Once retrieved, data is temporarily stored in the control's buffer.

In unbound mode, the control uses the **ssRowBuffer** object to populate the grid. You use the **UnboundReadData** event procedure to fill the RowBuffer with data. As long as the RowBuffer is filled with the number of rows of data specified by the **RowCount** property, the control will continue to generate the **UnboundReadData** event. However, the RowBuffer will not request more than a maximum of ten rows of data at a time.

When all of the data has been read into the control, you signal to the control that it should stop reading data by setting the **RowCount** property of the RowBuffer object to a value less than the value of the **RowCount** property. The control will then stop generating 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 the prior **RowCount** value. At this point the control has read all the data and knows exactly how many rows there are, and will adjust itself accordingly.

See Also

UnboundAddData event, UnboundDeleteRow event, UnboundPositionData event, UnboundWriteData event, ssRowBuffer Object, Performance Tuning

Example

Exercise 2 and Exercise 3 of the Data Grid Guided Tours contain complete code examples of how to use this event. Examples include using Unbound mode to connect the Grid to a Recordset object and using Unbound mode to connect the Grid to a memory array.

UnboundWriteData Event

Applies To

SSDBGrid

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:

Parameter	Description
RowBuf	The ssRowBuffer object that contains the modified data.
WriteLocation	A value identifying the unique bookmark of the row as specified in the **UnboundReadData** and **UnboundWriteData** 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.

In the ssRowBuffer, fields that have not had their value changed are Null.

To cancel the **UnboundWriteData** event, set the **RowCount** property of the **ssRowBuffer** object to 0.

This event is not generated if the update is canceled in the **BeforeUpdate** event procedure.

See Also

UnboundAddData event, UnboundDeleteRow event, UnboundPositionData event, UnboundReadData event, UnboundWriteData event, ssRowBuffer Object

Example

Exercise 2 and Exercise 3 of the Data Grid Guided Tours contain complete code examples of how to use this event. Examples include using Unbound mode to connect the Grid to a Recordset object and using Unbound mode to connect the Grid to a memory array.

Underline Property

Applies To

Font object, Headfont object

Description

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

Syntax

object . **Underline**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying the font style as described in Settings.

Settings

Setting	Description
True	Turns on underline formatting.
False	(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.

See Also

Font object, Headfont object

Example

The following sample code displays the caption text with underline:

```
SSDBOptSet1.Font.Underline = True
SSDBOptSet1.Caption = "Caption"

SSDBGrid.HeadFont.Underline = True
```

Update Method

Applies To

SSDBGrid

Description

Updates any modified information in the grid. This method is applicable to all data modes.

Syntax

object . **Update**

Part	Description
object	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.

UpdateError Event

Applies To

SSDBGrid

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
ColIndex	The column of the current row that contains the error.
Text	The erroneous text.
ErrCode	The error code. The code represents a standard host environment error number.
ErrString	The error string. The string represents the actual error message.
Cancel	Set this to **True** 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."

See Also

IsCellValid

How the Data Grid handles data validation and error checking

UseDefaults Property

Applies To

SSDBGrid

Description

Determines whether the grid will use default values to populate new records.

Syntax

object . **UseDefaults**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether new records will be populated with default values, as described in Settings.

Settings

Setting	Description
True	(Default) The control will retrieve default values from the data source and use them to populate new records.
False	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.

UseExactRowCount Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Determines whether the grid portion scans for the exact row count.

Syntax

object . **UseExactRowCount**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether the exact row count is scanned, as described in Settings.

Settings

Setting	Description
True	(Default) Exact row count will be used.
False	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.

ValidateList Event

Applies To

SSDBCombo, SSDBDropDown

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**)

Part	Description
Text	The text to validate
RtnPassed	Set this to notify the control whether the data is valid or not. This parameter defaults to **True**, which indicates that the entered data is valid.

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 not valid, set *RtnPassed* to **False**.

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.

This event is generated when the **IsItemInList** method is invoked or when the **ListAutoValidate** property is set to **False** and one of the following occurs:

- For the SSDBCombo, the position of the current record in a bound recordset changes or if the **IsItemInList** method is invoked.
- For the SSDBDropDown, if the cell of the linked column in the SSDBGrid loses focus or either the **Update** or **IsItemInList** method is invoked.

ValidationError Event

Applies To

SSDBCombo, SSDBGrid

Description

Occurs when a masked edit field receives invalid input, as determined by the input mask.

Syntax

Sub *control*_**ValidationError(***InvalidText* **As String**, *StartPosition* **As Integer)**

The event parameters are:

Parameter	Description
InvalidText	A string value that evaluates to the text that caused the validation error.
StartPosition	An integer value that indicates the starting position of the offending string within the input string.

Remarks

InvalidText is the full text of the input string that caused the event, including the invalid character. This means that any placeholders and literal characters used in the input mask are included in *InvalidText*.

StartPosition is the position in *InvalidText* where the error occurred (the first invalid character). This is a zero-based value rather than a one-based value. For example, if the error occurs due to the first character entered, *StartPosition* will equal 0. *StartPosition* reflects the position that caused the event to fire, not necessarily the first character in the string that does not match the mask. For example, suppose the Data Grid reads a value from the database into a masked cell, and the first character of the data does not match the mask. If you place your cursor after the third character in the string and try to type an invalid character, the value returned by *StartPosition* will be 3 for the character you typed and not 0 for the existing invalid character.

In the Data Grid, the **ValidationError** event fires in response to a masking error in a particular Column object. To find out which column caused the error, use the **Col** property to return the current column. For example, if you wanted to clear the offending text, you could use the following code:

```
SSDBGrid1.Columns(SSDBGrid1.Col).Text = ""
```

See Also

ClipMode, Col, Mask, PromptChar, PromptInclude

Example

This event is called when invalid data is detected in a cell for which a mask has been set.

```
Private Sub SSDBGrid1_ValidationError(InvalidText As String, ByVal
StartPosition As Integer)

   MsgBox "The following data is incorrect: " & InvalidText & _
   vbCrLf & "The invalid data starts at character position " & _
   Str$(StartPosition + 1) & ".", vbOKOnly, "Formatting Error"

End Sub
```

Value Property (Bookmark Object)

Applies To

Bookmark object (SSDBData only)

Description

Sets or returns the value stored in a bookmark object.

Syntax

object . **Value**[= *value*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.

value	A variant expression specifying the value stored in a bookmark.

See Also

String

Example

This sample stores a value to a bookmark:

```
Dim vbookmark as variant
vBookmark = SSDBCommand1.SavedBookmark
SSDBData1.Bookmarks(0).Value = vBookmark
```

Value Property (Button Object)

Applies To

Button object

Description

Sets or returns the current state (checked / not checked) of the button object.

Syntax

object . **Value**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
value	A Boolean expression specifying whether the button is selected or not.

Settings

Setting	Description
True	(Default) Button object is selected.
False	Button object is not selected.

Value Property (SSDBGrid)

Applies To

ActiveCell Object, Column Object, SSDBCombo Control, ssRowBuffer Object

Description

Returns the value stored in a column for the current row.

Syntax

object . **Value**[= *value*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
Value	A variant expression specifying the value stored in a column.

Remarks

Value returns the value in its native data format while **Text** returns the value as a string.

As opposed to the **CellValue** method, this property returns the value currently in place, and not the value from the underlying data.

The **DataFieldList** and **DataFieldToDisplay** property settings of the SSDBCombo affect the value returned by the **Text** property.

See Also

CellText method, CellValue method, Text property

Example

The following code sets the value of column 0 in the current row of the grid to 5.

```
SSDBGrid1.Columns(0).Value = 5
```

VertScrollBar Property

Applies To

Column object

Description

Determines whether a vertical scrollbar is displayed in a column.

Syntax

object . **VertScrollBar**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether a vertical scrollbar is displayed, as described in Settings.

Settings

Setting	Description
True	Vertical scrollbar is displayed.
False	(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.

VisibleCols Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Returns the number of visible columns in the grid.

Syntax

object . **VisibleCols**

Part	Description
object	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.

See Also

Col, Cols

VisibleGrps Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Returns the number of visible groups in the grid.

Syntax

object . **VisibleGrps**

Part	Description
object	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 Property

Applies To

SSDBCombo, SSDBDropDown, SSDBGrid

Description

Returns the number of visible rows that could appear in the control's display area.

Syntax

object . **VisibleRows**[= *number*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the number of visible rows that could appear in the control's display area.

Remarks

VisibleRows is only available at runtime and is read-only.

The value returned by the **VisibleRows** property indicates how many visible rows could be displayed in the control's display area, and not how many actually do. For example, if eight rows could fit into the display area of the control but the control has only three rows of data, **VisibleRows** would return 8.

WhereIs Method

Applies To

SSDBData, SSDBOptSet, SSDBGrid

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**]**)**

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
X	Determines the X coordinate.
Y	Determines the Y coordinate.
scale	(Optional) Determines the scale to use as described in Settings.

Settings

The settings for *scale* are:

Setting	Description
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:

Value	Description
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:

Value	Description
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:

Value	Description
0	Pointing at nothing
1	Pointing at button
2	Pointing at caption

There are constants for the settings of this parameter.

See Also

ButtonFromCaption, ButtonFromPos

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
```

WidthGap Property

Applies To

SSDBOptSet

Description

Returns or sets the minimum horizontal distance between columns.

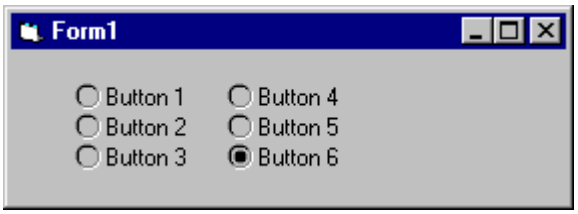
Syntax

object . **WidthGap**[= *number*]

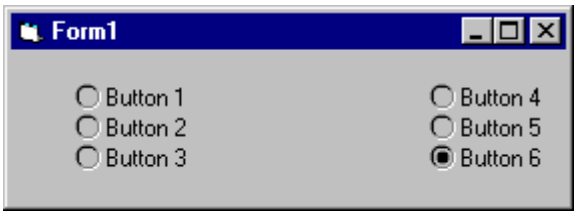
Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
number	An integer expression specifying the minimum amount of horizontal distance between columns.

Remarks

The following example shows a DataOptionSet with a small **WidthGap** setting:



The following example shows a DataOptionSet with a large **WidthGap**:



See Also

ColOffSet, RowOffSet

WordWrap Property

Applies To

SSDBCommand, SSDBOptSet, SSDBData

Description

Determines if the text specified in the **Caption** property will be wrapped.

Syntax

object . **WordWrap**[= *boolean*]

Part	Description
object	An object expression that evaluates to an object or a control in the Applies To list.
boolean	A Boolean expression specifying whether to wrap the caption text, as described in Settings.

Settings	Description
True	(Default) Caption text will wrap to multiple lines as needed.
False	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 an Enhanced Data Control **without wordwrap**:

The following is an example of an Enhanced Data Control **with wordwrap**:

See Also

MultiLine

Data Widgets Constants

Alignment Constants

Constant	Value	Description
ssAlignToRight	0	Right justify
ssAlignToLeft	1	Left justify

AlignmentButtonPicture Constants

Constant	Value	Description
ssPic AlignLeft	0	Left
ssPicAlignRight	1	Right
ssPicAlignLeftOfText	2	Left of text
ssPicAlignRightOfText	3	Right of text
ssPicAlignFitToCaption	4	Fit to caption
ssPicAlignTile	5	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

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

CellNavigation Constants

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

ColumnCaptionAlignment Constants

Constant	Value	Description
ssColCapAlignLeftJustify	0	Left justify
ssColCapAlignRightJustify	1	Right justify
ssColCapAlignCenter	2	Center
ssColCapAlignUseColumnAlignment	3	Use alignment specified for Column cells

ClipMode Constants

Constant	Value	Description
ssIncludeMaskLiterals	0	Mask literals will be copied to the clipboard
ssExcludeMaskLiterals	1	Mask literals will not be copied to the clipboard

ClippingOverride Constants

Constant	Value	Description
ssClippingOverrideAuto	0	Automatic clipping based on printer driver
ssClippingOverrideYes	1	Manual clipping
ssClippingOverrideNo	2	No clipping

Criteria Constants

Constant	Value	Description
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

Constant	Value	Description
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

DriverOverride Constants

Constant	Value	Description
ssDriverOverrideAuto	0	Automatic driver detection & override
ssDriverOverrideYes	1	Manual override of printer driver
ssDriverOverrideNo	2	No override of printer driver

ExportFlags Constants

Constant	Value	Description
ssExportCurrentRow	1	Export current row
ssExportSelectedRows	2	Export all selected rows
ssExportNonSelectedRows	4	Export all non-selected rows
ssExportAllRows	7	Export all rows (Combines 1, 2 & 4)
ssExportFieldNames	16	Include field names
ssExportColumnHeaders	32	Include column headers
ssExportInLayoutOrder	64	Export in order specified by grid layout
ssExportHiddenColumns	128	Include hidden columns
ssExportOverwriteExisting	256	Overwrite existing output file
ssExportAppendToExisting	512	Append to existing output file

ExportType Constants

Constant	Value	Description
ssExportTypeText	0	Text file
ssExportTypeHTMLTable	1	All rows in single HTML file
ssExportTypeHTMLRowFiles	2	Single rows in multiple HTML files

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

PrintColHeaders Constants

Constant	Value	Description
ssNoColumnHeaders	0	No column headers
ssUseCaption	1	Use caption
ssUseFieldName	2	Use field name

PrintFlags Constants

Constant	Value	Description
ssPrintCurrentRow	1	Print current row
ssPrintSelectedRows	2	Print selected rows
ssPrintNonSelectedRows	4	Print non-selected rows
ssPrintAllRows	7	Print all rows (combines 1, 2 & 4)
ssPrintFieldOrder	16	Print in field order (must use with another flag or flags)

PrintGridLines Constants

Constant	Value	Description
ssPrintGridLinesNone	0	Do not print grid lines
ssPrintGridLinesColGrpHeadsOnly	1	Print grid lines around headers
ssPrintGridLinesRowsOnly	2	Print grid lines around rows
PrintGridLinesAll	3	Print all grid lines

PrintHeaders Constants

Constant	Value	Description
ssTopOfPage	0	Top of page
ssTopOfReport	1	Top of report

PrintPageBreakGroups Constants

Constant	Value	Description
ssPrintPageBreakOnColumns	0	Break page on columns. Split groups as needed.
ssPrintLeftAlignGroups	1	Break page on groups. Left align groups on page.
ssPrintCenterGroups	2	Break page on groups. Center groups on page.

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 movment using arrow keys
ssRowNavigationLRLock	1	Left/Right movement only within row
ssRowNavigationUDLock	2	Up/Down movement only within row
ssRowNavigationAllLock	3	All movment 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

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* / **Error Text** / Description
30402	*ssItemNotInList* **"Item is not in list"** Validation determined that the current edit text is not in the associated dropdown list.
30403	*ssItemNotInList* **"Item is not in list"** Validation determined that the current edit text is not in the associated dropdown list.
30422	*ssItemNotInList* **"Item is not in list"** Validation determined that the current edit text is not in the associated dropdown list.
30423	*ssNullEdit* **"A null value is not allowed for this field"** Validation determined that the current edit text is null, but nulls are not allowed.

30424	*ssBadTypeEdit*
	"The value is not of the correct type for this field"
	Validation determined that the data type of the value currently in edit text does not match the data type of the associated list.
30425	*ssBitMap*
	"PictureDropDown must be a bitmap"
	You tried setting the PictureDropDown property to something other than a bitmap
30426	*ssDropItemsRange*
	"Value must be from 1 to 100"
	You tried setting the MaxDropDownItems or MinDropDownItems outside the allowed range.
30427	*ssMaxLessMinDropItem*
	"MaxDropDownItems cannot be less than MinDropDownItems"
	You tried setting MaxDropDownItems to a number smaller than the MinDropDownItems value.
30428	*ssdatbMinMoreMaxDropItem*
	"MinDropDownItems cannot be greater than MaxDropDownItems"
	You tried setting the MinDropDownItems to a value greater than the MaxDropDownItems.
30430	*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 FieldDelimeter 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.

30454 *ssPrintErrorInternal*

"An internal printing error has occurred."

Printing could not continue due to a problem with the program or the control, such as a data source failure.

30456 *ssPrintErrorDLLMissing*

"Printing support file ssprn32.dll missing or not registered."

"Printing support file ssprn16.dll missing or not registered."

An essential 32-bit (or 16-bit) system file was not properly installed on the user's system.

30457 *ssPrintErrorUserCancelled*

"Print job cancelled by user."

The user manually stopped the print job before it was complete.

30467 *ssBindingErrorMultipleSources*

"The control can only be bound to one type of data source at a time."

You tried to set the control's data source to an ODBC provider, then to an OLE DB provider, or vice versa.

30468 *ssBindingErrorDllMissing*

"OLE DB Support file ssr2c.dll missing or not registered."

The required OLE DB support file, SSR2C.DLL, was either not installed or not registered on the distribution system.

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	*ssEDCAlignment*
	"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.

HTML Template Codes

Below is a list of the valid attributes of the {SSREPLACE} token, with descriptions of how each attribute operates. Attributes fall into the following general categories:

- **Data Attributes**
- **Formatting Attributes**
- **Table Formatting Attributes**
- **Hyperlink Attributes**
- **Template Samples**

Introduction

When exporting HTML, the **Export** method can take an *HTMLTemplate* parameter which must be the filename of an existing HTML template file. This file can contain any valid HTML tags and must have one or more {SSREPLACE} tokens.

When Data Widgets uses a template file to generate HTML, it simply scans the template looking for {SSREPLACE} tokens. If it encounters a token, it will replace it with the grid data specified by the token's atributes. Any other text is simply passed through to the generated file.

The attributes of the token determine what will be inserted into the generated file in place of the token. The most important attributes are the DATA attribute, the TABLE attribute and the FIELD attribute. Combining DATA and TABLE causes Data Widgets to replace the token with the entire structure of the Data Grid and to fill it with the data from the grid. Combining the DATA and FIELD attributes causes Data Widgets to replace the token with the data from the specified field in the grid. The TABLE and FIELD attributes are mutually exclusive within a single {SSREPLACE} token.

By specifying other attributes, you can replace the token with other information from the grid, such as the grid's caption or the value of its **TagVariant** property. The CAPTION and TAGVAR attributes will accomplish this. Because they cause the token to be replaced with something other than grid data, the CAPTION, TAGVAR and FIELD attributes are mutually exclusive within the same {SSREPLACE} token.

When generating a complete table using the TABLE attribute, some attributes work differently than they would if they were used alone or with the FIELD attribute. For example, {SSREPLACE CAPTION TAGVAR} is an invalid token, becuase the token can only be replaced with either the grid's caption or its TagVariant, not both. However, {SSREPLACE DATA TABLE CAPTION TAGVAR} is a legitimate token. It tells Data Widgets to replace the token with a table containing the grid data, and to include an extra row for the grid caption and another extra row for the TagVariant. See the TABLE data attribute for a complete list of the attributes that work differently when paired with TABLE.

Data Attributes

The replacement token must begin with "{SSREPLACE" and end with "}". Between these two can be any of the following attributes.

CAPTION	Replaced with the grid's Caption property. **Note** CAPTION, TAGVAR and FIELD are mutually exclusive within a single SSREPLACE token that does not contain the TABLE attribute. CAPTION may be combined with other attributes in a table-generating token.
COLHEAD	Replaced with this field's Column.Caption property (or all column Captions if FIELD=*). Requires the FIELD attribute or the TABLE attribute.
COLTAGVAR	Replaced with this field's Column.TagVariant property (or all column TagVariants if FIELD=*). Requires the FIELD attribute or the TABLE attribute.
DATA	Replaced with this field's data value for a single row (or all field's data if FIELD=*). Requires the FIELD attribute or the TABLE attribute. **Note** If this is an *ssExportTypeHTMLTable* export and the token appears inside one or more <TD> or <TH> tags inside an HTML table (or is included with the TABLE attribute), all the table row(s) from <TR> through </TR> will be repeated for each row in the export.
FIELD	Identifies the field in the grid by name (e.g. FIELD=Au_ID) or by ordinal position within the Columns collection (e.g. FIELD=0) or * for all fields. All fields has meaning only inside the <TD> or <TH> tags inside an HTML table. If the field name contains embedded spaces, enclose it in brackets [ ], for example FIELD=[year born]. **Note** TABLE and FIELD are mutually exclusive within a single SSREPLACE token. CAPTION, FIELD and TAGVAR are mutually exclusive within a single SSREPLACE token.
FLDNAME	Replaced with this field's Column.DataField property (or all field names if FIELD=*). Requires the FIELD attribute or the TABLE attribute.
GRPHEAD	Replaced with this field's Group.Caption property (or all group captions if FIELD=*). Requires the FIELD attribute or the TABLE attribute.
GRPTAGVAR	Replaced with this field's Group.TagVariant property (or all group TagVariants if FIELD=*). Requires the FIELD attribute or the TABLE attribute.
LEVELS	This only has meaning if FIELD=* or TABLE is specified. It means that if the grid's **LevelCount** property is greater than 1 then the column headings and data will conform to the grid's layout and span multiple rows in the table for each logical row in the grid. The COLSPAN attribute (described under Table Formatting Attributes) is automatically implied with LEVELS.
TABLE	Replaced with the entire grid as an HTML table. If the SSREPLACE token is not contained within a <TABLE></TABLE> pair then one will be generated by default. **Note** If TABLE is specified, one or more of GRPHEAD, COLHEAD, FLDNAME and/or DATA must be specified. TABLE and FIELD are mutually exclusive within a single SSREPLACE token.

CAPTION, TAGVAR, GRPTAGVAR and COLTAGVAR are optional attributes that may be specified with the TABLE attribute. Including one or more of these in the same token as the TABLE attribute will cause the specified information to be included in the generated table.

If multiple attributes are specified, the rows will appear in the HTML table in the following order:

1 - CAPTION	1 row (containing a single cell)
2 - GRPHEAD	1 row
3 - COLHEAD	1 row (or 1 row for each level if LEVELS is specified)
4 - FLDNAME	1 row (or 1 row for each level if LEVELS is specified)
5 - DATA	1 row (or 1 row for each level if LEVELS is specified) for each selected output row
6 - COLTAGVAR	1 row (or 1 row for each level if LEVELS is specified)
7 - GRPTAGVAR	1 row
8 - TAGVAR	1 row (containing a single cell)

TAGVAR — Replaced with the grid's TagVariant property.

Note CAPTION, TAGVAR and FIELD are mutually exclusive within a single SSREPLACE token that does not contain the TABLE attribute. TAGVAR may be combined with other attributes in a table-generating token.

Note CAPTION, COLHEAD, COLTAGVAR, DATA, FLDNAME, GRPHEAD, GRPTAGVAR, and TAGVAR are mutually exclusive within a single SSREPLACE token except when included with the TABLE attribute.

Formatting Attributes

The following attributes generate formatting information for the replaced text. Note that these attributes simply enable the formatting of the original grid to be reflected in the HTML - they do not actually format the output. For example, including the "B" token will cause data to be displayed in boldface only if it is boldfaced in the Data Grid. Otherwise, data will use whatever HTML formatting is specified for the {SSREPLACE} token.

Unfortunately, there is no way of optimizing these settings in an HTML table, so they will be generated for each cell in the table.

B — Places <B></B> around the replaced text if that text (in the grid) was bold.

COLOR — Places <FONT COLOR=...></FONT> around the replaced text with the forecolor used for that text in the grid.

EM — Places <EM></EM> around the replaced text if that text (in the grid) was italic.

FACE — Places <FONT FACE=...></FONT> around the replaced text with the face name of the font used for that text in the grid.

I — Places <I></I> around the replaced text if that text (in the grid) was italic.

Note If both the EM and I attributes are specified only the <I></I> tags will be generated.

SIZE — Places <FONT SIZE=...></FONT> around the replaced text with a number between 1 and 7 depending on the point size of the font used for that text in the grid based on the following:

- Point sizes greater than 24 are exported as Size="7"
- Point sizes greater than 18 and less than or equal to 24 are exported as Size="6"
- Point sizes greater than 14 and less than or equal to 18 are exported as Size="5"
- Point sizes greater than 12 and less than or equal to 14 are exported as Size="4"
- Point sizes greater than 10 and less than or equal to 12 are exported as Size="3"
- Point sizes greater than 8.5 and less than or equal to 10 are exported as Size="2"
- Point sizes less than or equal to 8.5 are exported as Size="1"

STRIKE — Places <STRIKE></STRIKE> around the replaced text if that text (in the grid) was strikethru.

STRONG	Places <STRONG></STRONG> around the replaced text if that text (in the grid) was bold. **Note** If both the STRONG and B attributes are specified only the <B></B> tags will be generated
U	Places <U></U> around the replaced text if that text (in the grid) was underlined.

Note If FACE, SIZE and COLOR are all specified, only one <FONT ..></FONT> wrapper is generated.

Table Formatting Attributes

The following attributes generate formatting information which is exported as HTML attributes that go with the <TABLE>, <CAPTION>,<TR>,<TH>, <TD> or <P> tags that enclose the {SSREPLACE} token. These settings are optimized for HTML table cells and rows. For example if the BGCOLOR of most cells in a row is the same color, then that BGCOLOR attribute is added to the row's <TR> tag and the cells which are exceptions will have a different BGCOLOR attribute added to their <TH> or <TD> tags.

ALIGN	Adds a align=left, right or center attribute to the tag depending on the alignment of that text in the grid. Only applies to <TH> and <TD> tags.
BGCOLOR	Adds a bgcolor=#.... to the tag containing the backcolor of that text in the grid. Only applies to <TABLE>, <CAPTION>,<TR>,<TH> and <TD> tags.
COLSPAN	Adds a colspan=.. to the tag containing the number of columns to span. Only applies to <TH> and <TD> tags and is ignored if the GRPHEAD, CAPTION or LEVELS attributes are not specified. The value of colspan is affected by whether or not LEVELS was specified (see above).
WIDTH	Adds a width=... to the tag containing the width (in pixels) of that column or group in the grid. Only applies to <TH> and <TD> tags. If FIELD=* or TABLE was specified (see above) the width attributes are not added to each cell. Instead, an extra row is appended to the table with multiple <TD WIDTH="..."></TD> pairs. This extra row is invisible in Microsoft Internet Explorer but shows up as a minimum height (approximately 8 pixels) blank row in Netscape Navigator. **Note** WIDTH is normally only useful when you have a complicated grid layout with multiple levels (LevelCount > 1).

Hyperlink Attributes

The following attributes are used to automatically generate hyperlink anchor tags and have meaning only for *ssExportTypeHTMLTable* type exports and if the DATA attribute is also specified.

LINKANCHOR	Required (and only has meaning) if FIELD=* or TABLE is also specified. This identifies the field in the exported table by name (e.g. LINKANCHOR =Author) or by ordinal position within the Columns collection (e.g. LINKANCHOR =1) whose value will be wrapped by the anchor tag.
LINKFIELD	Required for generating an anchor tag. Identifies the field in the grid by name (e.g. LINKFIELD=Au_ID) or by ordinal position within the Columns collection (e.g. LINKFIELD=0) whose value will be used for constructing the HREF= attribute value for each exported row.
LINKMASK	Optional. If specified will be used along with the value of the LINKFIELD to generate the HREF attribute. For example if LINKMASK=id.htm and the value of the LINKFIELD column is 1001 for an exported row the anchor tag would be <A HREF="id1001.htm"></A>. If LINKMASK isn't specified an extension of ".htm" is generated by default..
LINKTARGET	Optional. This specifies the value of the TARGET attribute of the anchor tag so that the hyperlink can load the .htm file into a named frame.

Template Samples

1. The following will replace the value of Au_ID (including its formatting information) inside the <P></P> tags for the single row being exported in this file. This would only make sense for an export type of *ssExportTypeHTMLRowFiles*.

```
<HTML>
<P>{SSREPLACE FIELD=[au_id] DATA FACE B U EM STRIKE
SIZE COLOR BGCOLOR ALIGN}</P>
</HTML>
```

2. The following will export all of the grid's column headings in the first row of the HTML table (each column heading will be wrapped inside a <TD></TD> pair. Then the same thing would be done for the data on the next and subsequent rows. Each row of data selected for export would be wrapped inside a <TR></TR> pair. This would only make sense for an export type of *ssExportTypeHTMLTable*.

```
<HTML>
<TABLE>
<TR>
<TD>{SSREPLACE FIELD=* COLHEAD FACE B U EM STRIKE
SIZE COLOR BGCOLOR ALIGN}</TD>
</TR>
<TR>
<TD>{SSREPLACE FIELD=* DATA FACE B U EM STRIKE
SIZE COLOR BGCOLOR ALIGN}</TD>
</TR>
</TABLE>
</HTML>
```

3. The following will replace the entire table (as it appears in the grid) including the grid's caption, group headings, column headings and data. In addition, the 'Author' column would display a hyperlink. It would be wrapped in an anchor tag which would look like '<A HREF="idxxxx.htm" TARGET="DetailFrame"></A>'. Where xxxx would be contain the Au_ID value for that row. This would only make sense for an export type of *ssExportTypeHTMLTable*.

```
<HTML>
{SSREPLACE TABLE CAPTION GRPHEAD COLHEAD DATA FACE B U EM
STRIKE SIZE COLOR BGCOLOR ALIGN WIDTH LEVELS LINKANCHOR=Author
LINKFIELD=Au_ID LINKMASK=id.htm LINKTARGET=DetailFrame}
</HTML>
```

Note Although you can specify multiple {SSREPLACE} tokens in an HTML template, you will not get multiple copies of the same data. For example, if your HTML template contains the following two lines:

```
{SSREPLACE TABLE DATA CAPTION GRPHEAD COLHEAD}
{SSREPLACE TABLE DATA CAPTION GRPHEAD COLHEAD}
```

you will not see two identical tables in the generated HTML file. This is as you would expect; after exporting all the grid data using the first {SSREPLACE} token, there is no data left to export by the second token. Instead, you will see a second grid with only group and column headers.

If you want to include the same data twice in the same HTML file, you must invoke the **Export** method twice, specifying that the second export should be appended to the file created by the first export.

Technical Reference

System Requirements

You must have the following to utilize Data Widgets:

- Microsoft Visual Basic version 4 or higher, or another development tool that supports ActiveX 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, Windows 98, or Windows NT 4.0 or later.
- For the 16-bit version of Data Widgets, you must have Windows version 3.1 or later, running in enhanced mode.

Included Files

The Setup program will place OCX files in the directories you have specified. Sample applications 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.

Filename(s)	Description
ASYCFILT.DLL	Support DLL
CTL3DV2.DLL	16-bit support DLL
COMCAT.DLL	Support DLL
COMDLG16.OCX	16-bit Common Dialog OCX (used for demo programs)
COMDLG32.OCX	32-bit Common Dialog OCX (used for demo programs)
COMPOBJ.DLL	Support DLL
DAO2516.DLL	Support DLL
DATW1TO3.HLP	Version 1 to 2 Conversion help file.
DATW3FAQ.HLP	Frequently Asked Questions help file.
DW2TO316.EXE	16-bit version conversion utility.
DW2TO332.EXE	32-bit version conversion utility.
MFC40.DLL	Support DLL (Microsoft Foundation Class DLL)
MFC42.DLL	Support DLL (Microsoft Foundation Class DLL)
MFCO40.DLL	Support DLL (Microsoft Foundation Class DLL)
MFCOLEUI.DLL	Support DLL (16-bit 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)
MSOUTL32.OCX	32-bit Outline Control OCX (used for demo programs)
MSVBVM50.DLL	Visual Basic 5 Runtime DLL
MSVCRT.DLL	Support DLL (Microsoft VC DLL)

MSVCRT20.DLL	Support DLL (Microsoft VC DLL)
MSVCRT40.DLL	Support DLL (Microsoft VC DLL)
OC25.DLL	Support DLL (16-bit data binding DLL)
OLE2.DLL	Support DLL (OLE DLL)
OLE2DISP.DLL	Support DLL (OLE DLL)
OLE2NLS.DLL	Support DLL (OLE DLL)
OLEAUT32.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.HTM	Data Widgets late-breaking information
SCP.DLL	Support DLL
SELECT.BMP	SSDBGrid row selector bitmap
SSCMDP16.EXE	16-bit SSDBCommand custom property pages
SSCMDP32.EXE	32-bit SSDBCommand custom property pages
SSCMDPO.EXE	OLE DB SSDBCommand custom property pages
SSDW3A32.CAB	CAB file
SSDW3B32.CAB	CAB file
SSDW3A32.DEP	Dependency file
SSDW3A32.DEP	Dependency file
SSDW3A16.OCX	16-bit OCX containing SSDBData, SSDBOptSet, SSDBCommand
SSDW3A32.OCX	32-bit OCX containing SSDBData, SSDBOptSet, SSDBCommand
SSDW3AO.CAB	CAB File
SSDW3AO.DEP	Dependency file
SSDW3AO.OCX	32-bit OCX with OLE DB support containing SSDBData, SSDBOptSet, SSDBCommand
SSDW3B16.OCX	16-bit OCX containing SSDBGrid, SSDBDropDown, SSDBCombo
SSDW3B32.OCX	32-bit OCX containing SSDBGrid, SSDBDropDown, SSDBCombo
SSDW3BO.CAB	CAB File
SSDW3BO.DEP	Dependency file
SSDW3BO.OCX	32-bit OCX with OLE DB support containing SSDBGrid, SSDBDropDown, SSDBCombo
SSDATWD3.CNT	Data Widgets Online Help Contents File
SSDATWD3.HLP	Data Widgets Online Help
SSDBHPIC.BMP	SSDBData button bitmap
SSDBHPC2.BMP	SSDBData button bitmap
SSDODEMO.EXE	DataOptionSet demo
SSDOSP16.EXE	16-bit SSDBOptSet custom property pages
SSDOSP32.EXE	32-bit SSDBOptSet custom property pages
SSDOSPO.EXE	OLE DB SSDBOptSet custom property pages

SSEDCP16.EXE	16-bit SSDBData custom property pages
SSEDCP32.EXE	32-bit SSDBData custom property pages
SSEDCPO.EXE	OLE DB SSDBData custom property pages
SSGRID16.EXE	16-bit SSDBGrid Layout Editor
SSGRID32.EXE	32-bit SSDBGrid Layout Editor
SSMEDT16.DLL	16-bit masked edit DLL
SSMEDT32.DLL	32-bit masked edit DLL
SSPP16.DLL	16-bit Property Pages DLL
SSPP32.DLL	32-bit Property Pages DLL
SSPRN16.DLL	16-bit printer DLL
SSPRN32.DLL	32-bit printer DLL
SSR2C.DLL	32-bit OLE DB DLL
STDOLE.TLB	Support File
STDOLE2.TLB	Support File
STORAGE.DLL	Support DLL (System DLL)
TABCTL16.OCX	16-bit Tab Control OCX (used for demo programs)
TABCTL32.OCX	32-bit Tab Control OCX (used for demo programs)
THREED16.OCX	16-bit 3D Control OCX (used for demo programs)
THREED32.OCX	32-bit 3D Control OCX (used for demo programs)
TYPELIB.DLL	Support DLL (type library DLL)
VAEN2.DLL	Support DLL
VBAJET.DLL	Support DLL (compatibility layer DLL)
VB4EN16.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 the tutorials.

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.

Technical Support

World Wide Web

The Sheridan Software World Wide Web site (www.shersoft™.com) 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

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 technical support representative or get answers using the Automated Fax Service.

Optimizing Data Widgets

The following tips give you some ideas for optimizing individual Data Widgets 3.1 controls. For more general information on improving application and control performance, see the section on **Performance Tuning**.

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 you 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.

For more information on optimizing the performance of the Data Widgets controls, see **Performance Tuning**.

Performance Tuning

The following are some helpful suggestions on how to improve the performance of your applications that use Data Widgets 3.1. The following information should help you solve some of the most common performance problems.

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 Data Grid 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 Data Grid 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 Data Combo or the Data Dropdown 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.

Solving Printing Problems

You may occasionally encounter unexpected results when using Data Widgets 3.1 to print information, particularly if your application is distributed across multiple Windows platforms using different types of hardware. If you encounter any printing problems, your first step should always be to check all the settings of the printer you are attempting to print to. Ensure that there is enough memory to handle the print job. Data Widgets reports can be printer memory-intensive, especially if you print using multiple fonts, with grid lines and shading. Try setting your printer driver to print TrueType fonts as graphics, or lowering the resolution of the printer to half its normal value (for example, from 600dpi to 300dpi.) You might also try simplifying the report so that it uses a single font and no graphics. Upgrading existing printer memory may be another option.

If you have eliminated printer memory considerations, there may be other causes for printing problems. Sheridan Software has tested Data Widgets 3.1's printing capabilities with an array of hardware devices and device drivers. We have found inconsistencies in the way certain printer drivers implement the printing APIs that Data Widgets uses to create its reports. Data Widgets 3.1 can detect the presence of these drivers and compensate automatically.

- If you are having problems with fonts extending outside their containers and overwriting grid lines, with text wrapping, or with blank borders around text, click here for more information on compensating by using the **ClippingOverride** property.

Note When printing reports, the height of a row is automatically adjusted to at least the height of the largest font used in the row. (It may also be greater, depending on the setting of the **RowAutoSize** property.) However, column and group headers *do not* automatically resize to accommodate larger fonts. You must manually adjust the height of the column and/or group headers in your grid so that the full caption can be seen, both on the screen and in the printout.

For other tips on optimizing the performance of the Data Widgets controls, see **Optimizing Data Widgets**.

Upgrading to Data Widgets 3

Converting DataWidgets 2 to 3

The new versions of the Data Widgets OCX controls have different Globally Unique Identifier (GUID) values than the 2 versions. To update your existing Visual Basic project files to version 3 of Data Widgets, you must change the project file to use the new identifier.

Note Make sure that Data Widgets 3 is properly installed on your system before upgrading any projects. To be safe, make backup copies of your Data Widgets 2 projects before making any changes.

It is also possible to upgrade existing Data Widgets 3.0 or 3.01 projects to use the OLE DB data binding features of version 3.1. This is a manual process, which is detailed below and also in the README.WRI file.

Upgrading Data Widgets 3 projects to OLE DB

If you want to convert a project using Data Widgets 2.0 bound to the DAO/ODBC Data Control or the ODBC Remote Data Control, follow the steps outlined below for upgrading a project from Data Widgets 2 to Data Widgets 3.1. Once your project is using Data Widgets 3.1 (ODBC-bound) you can change it to use the OLE DB-bound controls by following the procedure outline below.

1. Open up your VBP file using Windows Notepad or another text editor and add the following lines:

```
Object={B1C46850-3E6A-11d2-8FEB-00104B9E07A7}#3.0#0; ssdw3ao.ocx
Object={4A4AA691-3E6F-11D2-822F-00104B9E07A1}#3.0#0; ssdw3bo.ocx
```

   This line loads the Data Widgets OLE DB controls into your toolbox.

2. To remove the DAO/ODBC version of the controls from your toolbox, remove the following lines:

```
Object={8D650146-6025-11d1-BC40-0000C042AEC0}#3.0#0; ssdw3a32.ocx
Object={8D650141-6025-11D1-BC40-0000C042AEC0}#3.0#0; ssdw3b32.ocx
```

   (This step is optional. You are not required to remove the DAO/ODBC versions of the controls.)

3. Save the changes to your VBP file.

4. Again using your text editor, open each Visual Basic form (.FRM file) that uses one of the DAO/ODBC controls and do the following:

   - Change the following lines:

```
Object = "{8D650146-6025-11d1-BC40-0000C042AEC0}#3.0#0"; "ssdw3a32.ocx"
Object = "{8D650141-6025-11D1-BC40-0000C042AEC0}#3.0#0"; "ssdw3b32.ocx"
```

     to

```
Object = "{B1C46850-3E6A-11d2-8FEB-00104B9E07A7}#3.0#0"; "ssdw3ao.ocx"
Object = "{4A4AA691-3E6F-11D2-822F-00104B9E07A1}#3.0#0"; "ssdw3bo.ocx"
```

   - Then change any lines further down in the FRM file that refer to the DataWidgets DAO/ODBC controls. For example, if your form contained an Enhanced Data Control, you would change the following line:

```
Begin SSDataWidgets_A.SSDBData SSDBData1
```

to

```
Begin SSDataWidgets_A_OLEDB.SSOleDBData SSDBData1
```

If your form contained a DataGrid control, you would change the following line:

```
Begin SSDataWidgets_B.SSDBGrid SSDBGrid1
```

to

```
Begin SSDataWidgets_B_OLEDB.SSOleDBGrid SSDBGrid1
```

4. Save the changes to the .FRM file.

Once you have saved the changes to the FRM file, you can load your project into Visual Basic 6 and bind the Grid/Combo/Dropdown to either a DataEnvironment or an ADO Data Control as per the Visual Studio documentation.

Note that you may receive an error when you first load forms converted in this way. If you check the log file, it will state:

"*Property Bindings in SSDBGrid1 could not be set. "*

All this means is that the OLE DB controls could not be bound to the ODBC DataSource defined in the FRM file. Simply change the **DataSource** property of the offending controls to their OLE DB equivalents.

Automatic Upgrading of Data Widgets 2 to 3.X

Sheridan Software realizes that you may want to continue using versions 2 and 3 of Data Widgets simultaneously. For example, you might have multiple applications in a production environment that use version 2. You may want to make changes to the code of these applications without upgrading them to the new versions of the Data Widgets controls. For this reason, Data Widgets 3 has been constructed so that Visual Basic does not automatically upgrade your Data Widgets 2 applications when you open them in the development environment.

Because Visual Basic's automatic upgrading capability is not implemented, Version 3 includes a stand-alone upgrade utility that gives you the ability to selectively upgrade Visual Basic project files on your hard disk from version 2 to 3. The utility comes in both 16-bit and 32-bit versions, and allows you to search for project files across multiple drives and directories. You can also add projects to the utility manually via a standard File Open dialog.

The files that have been located or added appear in a list box with check marks next to them. You can then manually select or de-select any of the files on the list. Once you have completed your file selection, you click the "Convert" button and all the specified files will be searched for references to Data Widgets 2. If any are found, they will be upgraded to refer to version 3. If a project file does not contain a reference to Data Widgets 2, no action is taken on that file.

The names of the upgrade utility executables are `DW2TO316.EXE` (16-bit version) and `DW2TO332.EXE` (32-bit version.) They are located in the main Data Widgets installation subdirectory, which you specified during installation.

Note You must have the Data Widgets 3 controls installed on your machine in order to use the upgrade utility.

Manual Upgrading of Data Widgets 2 to 3.X

Use the following manual procedure to upgrade your 2 projects to 3:

1. Use a text file editor such as Notepad to open up the project (VBP) file you wish to upgrade.
2. Locate the line or lines that refers to the Data Widgets 2 GUID values. They should look something like this:

```
Object={D981334D-B212-11CE-AE8C-0000C0B99395}#2.0#0; ssdata32.ocx
Object={BC496AED-9B4E-11CE-A6D5-0000C0BE9395}#2.0#0; ssdatb32.ocx
```

Depending on which controls your project uses, either or both lines may appear.

3. Replace the existing reference(s) to the 2 GUID value(s) with the following:

```
Object={8D650146-6025-11D1-BC40-0000C042AEC0}#3.0#0; ssdw3a32.ocx
Object={8D650141-6025-11D1-BC40-0000C042AEC0}#3.0#0; ssdw3b32.ocx
```

4. Save the project file. When you open the project in Visual Basic, the 3 versions of the controls should be in place.

No code changes are required to convert version 2 to version 3. These versions of the control use the same architecture and are 100% code-compatible. Of course, you must change your existing programs to take advantage of the newly added features, such as printing and data exporting.

If you need to update Visual C++ projects that contain Data Widgets 2, see the section on Upgrading Visual C++ Projects to Data Widgets 3.

Converting DataWidgets 1 to 3

Perhaps the most significant change from Data Widgets 1 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 of Data Widgets introduced 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, 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 will need to be modified to take advantage of the new structure. Additionally, many of the properties that existed in 1 have either been altered to conform with this new structure, or eliminated completely. To help ease the transition of users from 1 to 3, refer to the Online Help file 'DATW1TO3.HLP'.

Distributing Your Application

Once you have created a program using Data Widgets controls, you must distribute the following 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.

Filename	Description
SSDW3A16.OCX	16-bit OCX containing SSDBData, SSDBOptSet, SSDBCommand
SSDW3A32.OCX	32-bit OCX containing SSDBData, SSDBOptSet, SSDBCommand
SSDW3AO.OCX	32-bit OCX with OLE DB support containing SSDBData, SSDBOptSet, SSDBCommand. (Distribute *only* if your application uses OLE DB or ADO.)
SSDW3B16.OCX	16-bit OCX containing SSDBGrid, SSDBDropDown, SSDBCombo
SSDW3B32.OCX	32-bit OCX containing SSDBGrid, SSDBDropDown, SSDBCombo
SSDW3BO.OCX	32-bit OCX with OLE DB support containing SSDBGrid, SSDBDropDown, SSDBCombo.

	(Distribute *only* if your application uses OLE DB or ADO.)
SSMEDT16.DLL	16-bit DLL used for masked editing features
SSMEDT32.DLL	32-bit DLL used for masked editing features
SSPRN16.DLL	16-bit DLL used for printing features
SSPRN32.DLL	32-bit DLL used for printing features
SSR2C.DLL	32-bit DLL used for OLE DB and ADO database connectivity. (Distribute *only* if your application uses OLE DB or ADO.)

You may not need to distribute all of the preceding files. The files that are required will depend on the target platform (16-bit or 32-bit) and the functionality you use in your application. If your application uses the printing or data masking functions of Data Widgets 3.1, you must distribute the appropriate DLLs from the list above in addition to the OCX files. Similarly, if your application employs OLE DB or ADO database access, you must additionally distribute SSR2C.DLL.

Any system that runs your application must also have certain OLE-enabling support files installed. These are usually included with the operating system, but you may need to verify that the version numbers do not conflict. Also, there are some files included with Data Widgets that you may not re-distribute under the terms of your license agreement.

Non-Distributable Files

Non-distributable files are required to support the product in a development environment. Under the terms of your license agreement, you cannot distribute these files with your application. These include the executable and support files for the design-time environment and the design-time support files for product components.

The following files may **NOT** be distributed:

File	Description
SSCMDP16.EXE SSCMDP32.EXE SSDOSP16.EXE SSDOSP32.EXE SSEDCP16.EXE SSEDCP32.EXE SSGRID16.EXE SSGRID32.EXE SSPP16.DLL SSPP32.DLL	Property pages support files needed for development environment.
*.HLP *.CNT *.WRI	Any documentation files included with Data Widgets 3.1

This control is located in:

SSDW3A16.OCX, SSDW3A32.OCX, SSDW3AO.OCX

This control is located in:

SSDW3B16.OCX, SSDW3B32.OCX, SSDW3BO.OCX

Support files needed for distribution - 16 Bit

Due to the nature of the OLE architecture, Data Widgets 3.1 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 availablility 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 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 3.1 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.

File	Version
MFC42.DLL	6.00.8267.0
MSVCRT.DLL	6.00.8267.0
OLEAUT32.DLL	2.30.4261
OLEPRO32.DLL	5.0.4261

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
```

Appendix A: 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 help file.

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

Index