This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

Data Control

See Also Example Properties Methods Events

Provides access to data stored in databases using any one of three types of **Recordset** objects. The **Data** control enables you to move from record to record and to display and manipulate data from the records in bound controls. Without a **Data** control or an equivalent data source control like the **RemoteData** control, data-aware (bound) controls on a form can't automatically access data.

Syntax

Data

Remarks

You can perform most data access operations using the **Data** control without writing any code at all. Data-aware controls bound to a **Data** control automatically display data from one or more fields for the current record or, in some cases, for a set of records on either side of the current record. The **Data** control performs all operations on the current record.

If the **Data** control is instructed to move to a different record, all bound controls automatically pass any changes to the **Data** control to be saved in the database. The **Data** control then moves to the requested record and passes back data from the current record to the bound controls where it's displayed.

The **Data** control automatically handles a number of contingencies including empty recordsets, adding new records, editing and updating existing records, and handling some types of errors. However, in more sophisticated applications, you need to trap some error conditions that the **Data** control can't handle. For example, if the Microsoft Jet database engine has a problem accessing the database file, doesn't have permission, or can't execute the query as coded, a trappable error results. If the error occurs before your application procedures start or due to some internal errors, the Error event is triggered.

Bound Controls

The **DataList**, **DataCombo**, **DataGrid**, and **MSHFlexGrid** controls are all capable of managing sets of records when bound to a **Data** control. All of these controls permit several records to be displayed or manipulated at once.

The intrinsic **Picture**, **Label**, **TextBox**, **CheckBox**, **Image**, **OLE**, **ListBox** and **ComboBox** controls are also data-aware and can be bound to a single field of a **Recordset** managed by the **Data** control. Additional data-aware controls like the **MaskedEdit** and **RichTextBox** controls are available in the Professional and Enterprise Editions and from third-party vendors.

Operation

Once the application begins, Visual Basic uses **Data** control properties to open the selected database, create a **Database** object and create a **Recordset** object. The **Data** control's **Database** and **Recordset** properties refer to the newly created **Database** and **Recordset** objects which may be manipulated independently of the **Data** control with or without bound controls. The **Data** control is initialized *before* the initial Form_Load event for the form on which it is placed. If any errors occur during this initialization step a non-trappable error results.

When Visual Basic uses the Jet database engine to create a **Recordset**, no other Visual Basic operations or events can occur until the operation is complete. However, other Windows-based applications are permitted to continue executing while the

Recordset is being created. If the user presses CTRL+BREAK while the Jet engine is building a **Recordset**, the operation is terminated, a trappable error results, and the **Recordset** property of the **Data** control is set to **Nothing**. In design time, a second CTRL+BREAK causes Visual Basic to display the Debug window.

When you use a **Data** control to create a **Recordset** object or when you create a **Recordset** object in code and assign it to the **Data** control, the Microsoft Jet database engine automatically populates the **Recordset** object. As a result, bookmarks (and for snapshot-type **Recordset** objects, recordset data) are saved in local memory; the user doesn't need to manipulate the **Data** control, and you don't need to invoke the **MoveLast** method in code. Page locks used to create the **Recordset** are released more quickly, making it possible for other **Recordset** objects to access the same data. **Recordset** objects created in code but not assigned to the **Data** control aren't automatically populated by the Jet engine. Populate these objects through code. Because of the way that the **Data** control populates its **Recordset** in the background, an additional cloned **Recordset** might be created.

You can manipulate the **Data** control with the mouse, moving from record to record or to the beginning or end of the **Recordset**. The **EOFAction** and **BOFAction** properties determine what happens when the user moves to the beginning or end of a **Recordset** with the mouse. You can't set focus to the **Data** control.

Validation

Use the Validate event and the **DataChanged** property to perform last minute checks on the records being written to the database.

Data Access Objects

You can use the **Database** and **Recordset** data access objects created by the **Data** control in your procedures. The **Database** and **Recordset** objects each have properties and methods of their own, and you can write procedures that use these properties and methods to manipulate your data.

For example, the **MoveNext** method of a **Recordset** object moves the current record to the next record in the **Recordset**. To invoke this method, you could use this code:

```
Data1.Recordset.MoveNext
```

The **Data** control is capable of accessing any of the three types of Jet engine Version 3.0 **Recordset** objects. If you don't select a recordset type, a dynaset-type **Recordset** is created.

In many cases, the default type and configuration of the **Recordset** object created is extremely inefficient. That is, you might not need an updatable, fully-scrollable, keyset-type cursor to access your data. For example, a read-only, forward-only, snapshot-type **Recordset** might be far faster to create than the default cursor. Be sure to choose the most efficient **Type**, **Exclusive**, **Options** and **ReadOnly** properties possible for your situation.

Note The constants used to request a specific **Recordset** type when using the **Data** control are different than the constants used to determine the type of **Recordset** created or to create a **Recordset** using the **OpenRecordset** method.

To select a specific type of **Recordset**, set the **Data** control's **RecordsetType** property to:

Recordset Type	Value	Constant
Table	0	**vbRSTypeTable**
Dynaset	1	(Default) **vbRSTypeDynaset**
Snapshot	2	**vbRSTypeSnapshot**

Important The **Data** control cannot be used to access **Recordset** objects created with the **dbForwardOnly** option bit set.

Professional and Enterprise Editions

As far as data access is concerned, the primary difference between the Learning Edition, Professional and Enterprise Editions of Visual Basic is the ability to create new data access objects. In the Learning Edition, you can't declare (with the **Dim** keyword) variables as data access objects in code. This means that only the **Data** control can create **Database** and **Recordset** objects.

In the Professional and Enterprise Editions, you can create a new **Recordset** object and assign it to the **Data** control's **Recordset** property. Any bound controls connected to the **Data** control permit manipulation of the records in the **Recordset** you created. Make sure that your bound controls' **DataField** properties are set to field names that are valid in the new **Recordset**.

Stored Queries

Another important option when using the **Data** control is the ability to execute stored queries. If you create a **QueryDef** object beforehand, the **Data** control can execute it and create a **Recordset** using the **QueryDef** object's stored **SQL**, **Connect** and other properties. To execute a **QueryDef**, set the **Data** control's **RecordSource** property to the **QueryDef** name and use the **Refresh** method.

If the stored **QueryDef** contains parameters, you need to create the **Recordset** and pass it to the **Data** control.

BOF/EOF Handling

The **Data** control can also manage what happens when you encounter a **Recordset** with no records. By changing the **EOFAction** property, you can program the **Data** control to enter AddNew mode automatically.

You can program the **Data** control to automatically snap to the top or bottom of its parent form by using the **Align** property. In either case, the **Data** control is resized horizontally to fill the width of its parent form whenever the parent form is resized. This property allows a **Data** control to be placed on an MDI form without requiring an enclosing **Picture** control.

Using With Access 2000 Databases

The Data control can also be used to connect to Access 2000 databases. For details, see Intrinsic Data Control Is Usable with Access 2000 Databases.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DataBinding Object

See Also Example Properties Methods Events

The **DataBinding** object represents a bindable property of a component.

Syntax

DataBinding

Remarks

There is one **DataBinding** object for each property of a component marked as Bindable in the **Procedure Attributes** dialog box.

Visual Basic version 4.0 supported binding only one property of a control to a database at a time. Later versions of Visual Basic give you the ability to bind multiple properties of a control to a database. This is used most commonly with **User** controls. For more information on this, see Chapter 9 in "Creating ActiveX Components" in the *Component Tools Guide*.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DataBindings Collection

See Also Example Properties Methods Events

The DataBindings collection is an extender property that collects the bindable properties that are available to the developer and end-user.

Remarks

All bindable properties appear in the DataBindings collection at end user run time. At developer design time (control run time), only properties marked "Show in DataBindings collection at design time" will appear when the **DataBindings** property is accessed in the Properties window.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: DataCombo/DataList Controls

Visual Studio 6.0

DataCombo Control

See Also Example Properties Methods Events

The **DataCombo** control is a data-bound combo box that is automatically populated from a field in an attached data source, and optionally updates a field in a related table of another data source.

Syntax

DataCombo

Remarks

The **DataCombo** control is code-compatible with the **DBCombo** control. However the **DataCombo** control is optimized to work with ActiveX Data Objects (ADO).

Distribution Note The **DataCombo** control is found along with the **DataList** control in the Msdatlst.ocx file. To use this control in your application, you must add the .OCX file to the project. When distributing your application, install the Msdatlst.ocx file in the user's Microsoft Windows System or System32 directory. For more information on how to add an ActiveX control to a Visual Basic project, see "Standard ActiveX Controls."

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: DataGrid Control

Visual Studio 6.0

DataGrid Control

See Also Example Properties Methods Events

Displays and enables data manipulation of a series of rows and columns representing records and fields from a **Recordset** object.

Syntax

DataGrid

Remarks

The data-aware **DataGrid** control appears similar to the **Grid** control; however, you can set the **DataGrid** control's **DataSource** property to a **Data** control so that the control is automatically filled and its column headers set automatically from a **Data** control's **Recordset** object. The **DataGrid** control is really a fixed collection of columns, each with an indeterminate number of rows.

Each cell of a **DataGrid** control can hold text values, but not linked or embedded objects. You can specify the current cell in code, or the user can change it at run time using the mouse or the arrow keys. Cells can be edited interactively, by typing into the cell, or programmatically. Cells can be selected individually or by row.

If a cell's text is too long to be displayed in the cell, the text wraps to the next line within the same cell. To display the wrapped text, you must increase the cell's **Column** object's **Width** property and/or the **DataGrid** control's **RowHeight** property. At design time, you can change the column width interactively by resizing the column or by changing the column's width in the **Column** object's property page.

Use the **DataGrid** control's **Columns** collection's **Count** property and the **Recordset** object's **RecordCount** property to determine the number of columns and rows in the control. A **DataGrid** control can have as many rows as the system resources can support and up to 32767 columns.

When you select a cell, the **ColIndex** property is set, thus selecting one of the **Column** objects in the **DataGrid** object's **Columns** collection. The **Text** and **Value** properties of the **Column** object reference the contents of the current cell. The data in the current row can be accessed using the **Bookmark** property, which provides access to the underlying **Recordset** object's record. Each column of the **DataGrid** control has its own font, border, word wrap, and other attributes that can be set without regard to other columns. At design time, you can set the column width and row height and establish columns that are not visible to the user. You can also prevent users from changing the formatting at run time.

Note If you set any of the **DataGrid** column properties at design time, you will need to set all of them in order to maintain the current settings.

Note If you use the **Move** method to position the **DataGrid** control, you may need to use the **Refresh** method to force it to repaint.

The **DataGrid** control functions similarly to the **DBGrid** control except that it doesn't support an unbound mode.

Note This control is Unicode-enabled. When used on a Unicode-enabled system such as Microsoft Windows NT, the control passes Unicode data with no conversion. On other systems, however, data is converted from ANSI to Unicode and

back. For more information, see "ANSI, DBCS, and Unicode: Definitions" in the *Programmer's Guide*.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

DataGrid Object

See Also Example Properties Methods Events

Represents a virtual matrix containing labels and data points for the **MSChart** control.

Syntax

DataGrid

Remarks

The **DataGrid** object is configured as rows and columns. You can add and subtract rows, columns, and labels to this matrix to change the appearance of the chart.

© 2017 Microsoft

Visual Basic: MSChart Control

DataGrid Object Example

The following example sets the chart parameters for a three-dimensional bar chart, fills the chart with random data and labels the data grid columns.

```
Option Explicit
Option Base 1

Private Sub Command1_Click()
   Dim rowLabelCount, columnLabelCount, rowCount As Integer
   Dim columnCount, labelIndex, Column, Row As Integer
   MSChart1.ChartType = VtChChartType3dBar
   With MSChart1.DataGrid
   ' Set Chart parameters using methods.
      rowLabelCount = 2
      columnLabelCount = 2
      rowCount = 6
      columnCount = 6
      .SetSize RowLabelCount, columnLabelCount, _
      RowCount, ColumnCount
      ' Randomly fill in the data.
      .RandomDataFill
      ' Then assign labels to second Level.
      labelIndex = 2
      column = 1
      .ColumnLabel(column, labelIndex) = "Product 1"
      column = 4
      .ColumnLabel(column, labelIndex) = "Product 2"
      row = 1
      .RowLabel(row, labelIndex) = "1994"
      row = 4
      .RowLabel(row, labelIndex) = "1995"
   End With
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: DataCombo/DataList Controls

Visual Studio 6.0

DataList Control

See Also Example Properties Methods Events

The **DataList** control is a data-bound list box that is automatically populated from a field in an attached data source, and optionally updates a field in a related table of another data source.

Syntax

DataList

Remarks

The **DataList** control is code-compatible with the **DBList** control. However the **DataList** control is optimized to work with ActiveX Data Objects (ADO).

Distribution Note This **DataList** control is found along with the **DataCombo** control in the Msdatlst.ocx file. To use this control in your application, you must add the .OCX file to the project. When distributing your application, install the Msdatlst.ocx file in the user's Microsoft Windows System or System32 directory. For more information on how to add an ActiveX control to a Visual Basic project, see "Standard ActiveX Controls."

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DataMembers Collection

See Also Example Properties Methods Events

A collection of data members for a data source.

Syntax

DataMembers

Remarks

A data provider can have multiple sets of data that a data consumer can choose to bind to. Each set of data is called a "data member," and be either an **ADO Recordset**, a **Class** that implements the OLE Simple Provider (OSP) interface, or an OLEDB Provider created with Visual Basic. In any case, an arbitrary but unique string can be associated with the data member, and it is this identifying string that is added to the **DataMembers** collection using the **Add** method.

When configuring a data consumer to use a complex data source, you must set both the **DataSource** and the **DataMember** properties to fully qualify the data source. For example, if you configure a **TextBox** control to use the Data Environment as a data source, you must set its **DataSource** property to the Data Environment, and its **DataMember** property to a specific **Command** object. Conversely, when you create your own complex data source, the **DataMembers** collection allows your data source to serve multiple data sets to any data consumer.

For example, you can create a **User Control** configured as a data source by setting its **DataSourceBehavior** to **vbDataSource**. In the **Initialize** event, use the **Add** method to add the identifying strings of each data member to the **DataMembers** collection. Consequently, when the end user sets the **DataSource** property to your user control, and clicks **DataMembers** on the **Properties** window, those members added to the **DataMembers** collection will appear in the drop-down list.

Similarly, when creating a **Class** configured as a data source, in the **Initialize** event, invoke the **Add** method to add the identifying strings to the **DataMembers** collection. Then, to configure a data consumer to use the class, set its **DataSource** property to the class and its **DataMember** property to a member of the **DataMembers** collection.

When a data consumer's **DataSource** property is set to a class or User control configured as a data source, the **GetDataMember** event occurs. The event contains an argument, the *DataMember* argument, that passes the value of the **DataMember** property. The event also features an argument, the *Data* argument, that you can use to return the data to the consumer. In other words, in the GetDataMember event, query the *DataMember* value to determine which data member has been requested, and set the event's *Data* argument to the requested data source (i.e. an ADO Recordset, OLEDB provider, or class implementing the OSP interface).

A data member can also be an empty string. When programming the GetDataMember event, be sure handle this possibility by setting a default data member.

© 2017 Microsoft

Visual Basic Reference

BindingCollection Object, DataMembers Collection Example

The example uses a class module as a data source. When code to set the **DataSource** and **DataMember** properties of two **Binding** objects executes, the class module's Initialize event occurs; two ADO recordsets are created in that event, and the names of the recordsets are added to the **DataMembers** collection. The GetDataMember event and its arguments are used to return data to the data consumer.

To try the example, on the **Project** menu, click **References**, and set a reference to **Microsoft Data Binding Collection** and **Microsoft ActiveX Data Objects**. On the Project menu, click **Add Class Module**. Change the name of the class to MyDataClass, and set the **DataSourceBehavior** property to **vbDataSource**. Then draw two **TextBox** controls on a form. Paste the code into the **Form** object's code module.

```
Option Explicit
' Declare the object variables, one for a Class module named MyDataClass,
' and two more for each BindingCollection object one for each
' recordset).

Private clsData As New MyDataClass               ' Class module
Private bndColProducts As New BindingCollection  ' Bindings Collection
Private bndColSuppliers As New BindingCollection ' Bindings Collection

Private Sub Form_Load()
   ' Set DataSource and DataMember properties for each Bindings
   ' collection object.
   With bndColProducts
      .DataMember = "Products"
      Set .DataSource = clsData
      .Add Text1, "Text", "ProductName" ' Bind to a TextBox.
   End With

   With bndColSuppliers
      .DataMember = "Suppliers"
      Set .DataSource = clsData
      .Add Text2, "Text", "CompanyName" ' Bind to a TextBox.
   End With

   ' Change the Caption of Command1
   Command1.Caption = "MoveNext"

End Sub

Private Sub Command1_Click()
   clsData.MoveNext
End Sub
```

Paste the code below into the MyDataClass module. The **DataSourceBehavior** property must be set to **vbDataSource** in order to see the GetDataMember event. Run the project.

```
Option Explicit
' Declare object variables for ADO Recordset and Connection objects.
```

```
Private WithEvents rsProducts As ADODB.Recordset
Private WithEvents rsSuppliers As ADODB.Recordset
Private cnNwind As ADODB.Connection

Private Sub Class_Initialize()
   ' Add strings to the DataMembers collection.
   With DataMembers
      .Add "Products"
      .Add "Suppliers"
   End With

   ' Set Recordset objects.
   Set rsProducts = New ADODB.Recordset
   Set rsSuppliers = New ADODB.Recordset
   Set cnNwind = New ADODB.Connection

   ' Set the Connection object parameters.
   With cnNwind
      ' The Nwind.mdb that comes with Visual Basic must be installed on
      ' the computer or the code will fail. Otherwise alter the path to
      ' find the file on the computer.
      .Provider = "Microsoft.Jet.OLEDB.3.51"
      .Open "C:\Program Files\DevStudio\VB\Nwind.mdb"
   End With

   ' Open the recordset objects.
   rsSuppliers.Open "SELECT * FROM Suppliers", cnNwind, _
   adOpenStatic, adLockOptimistic
   rsProducts.Open "SELECT * FROM Products", cnNwind, _
   adOpenStatic, adLockOptimistic

End Sub

' The GetDataMember occurs when the DataSource property of a data
' consumer is set. In this case, the Bindings collection object is
' the consumer.
Private Sub Class_GetDataMember(DataMember As String, Data As Object)
   Select Case DataMember
   Case "Products"
      Set Data = rsProducts
   Case "Suppliers"
      Set Data = rsSuppliers
   Case ""
      ' Provide a default record source when no Data Member is specified.
      Set Data = rsProducts
   End Select

End Sub

Public Function MoveNext()
   If rsProducts.EOF Then
      rsProducts.MoveFirst
   Else
      rsProducts.MoveNext
   End If
End Function

Private Sub rsProducts_MoveComplete(ByVal adReason As _
ADODB.EventReasonEnum, ByVal pError As ADODB.Error, adStatus As _
ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
   ' Keep the two recordsets in sync. The first textbox displays
```

```
    ' the supplier of the product. If the SupplierID for both
    ' recordsets are equivalent, no change needed. Otherwise,
    ' move to first record and test for SupplierID. This example
    ' is for demonstration only as the method is not the most
    ' efficient.

    If rsSuppliers("SupplierID").Value = _
    pRecordset("SupplierID").Value Then Exit Sub

    rsSuppliers.MoveFirst
    Do While Not rsSuppliers.EOF
       If rsSuppliers("SupplierID").Value = _
       pRecordset("SupplierID").Value Then
          Exit Sub
       Else
          rsSuppliers.MoveNext
       End If
    Loop
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DataObject Object

See Also Example Properties Methods Events

The **DataObject** object is a container for data being transferred from an component source to an component target. The data is stored in the format defined by the method using the **DataObject** object.

Syntax

DataObject

Remarks

The **DataObject**, which mirrors the **IDataObject** interface, allows OLE drag and drop and clipboard operations to be implemented.

Most components support manual OLE drag and drop events, and some support automatic OLE drag and drop events.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DataObjectFiles Collection

See Also Example Properties Methods Events

A collection of strings which is the type of the **Files** property on the **DataObject** object.

Syntax

object.**DataObjectFiles**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Remarks

The **DataObjectFiles** collection is a collection of strings which represent a set of files which have been selected either through the **GetData** method, or through selection in an application such as the Windows Explorer.

Although the **DataObjectFiles** collection has methods and properties of its own, you should use the **Files** property of the **DataObject** object to view and manipulate the contents of the **DataObjectFiles** collection.

Here are some code examples showing the use of the **Files** property to view and manipulate data contained in the **DataObjectFiles** collection (where "Data" represents an object of type **DataObject**):

```
Debug.Print Data.Files(index)
For Each v in Data.Files
 Debug.Print v
Next v
Data.Files.Add "autoexec.bat"
Data.Files.Remove index
Data.Files.Clear
For i = 1 to Data.Files.Count
 Debug.print Data.Files(i)
Next i
```

Note This collection is used by the **Files** property only when the data in the **DataObject** object is in the **vbCFFiles** format.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DataObjectFiles Collection (ActiveX Controls)

See Also Example Properties Methods Events

A collection whose elements represent a list of all filenames used by a **DataObject** object (such as the names of files that a user drags to or from the Windows File Explorer.)

Syntax

object.**DataObjectFiles(***index***)**

The **DataObjectFiles** collection syntax has these parts:

Part	Description
object	An object expression that evaluates to a **DataObject** object.
index	An integer with a range from 0 to `DataObjectFiles.Count - 1`.

Remarks

Note This collection is used by the **Files** property only when the data in the **DataObject** object is in the **vbCFFiles** format.

The **DataObjectFiles** collection is used by the **Files** property to store filenames in a **DataObject** object. It includes the **Remove**, **Add**, and **Clear** methods which allow you to manipulate its contents.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

DataPoint Object

See Also Example Properties Methods Events

A member of the **DataPoints** collection that describes the attributes of an individual data point on a chart.

Syntax

DataPoint

Remarks

The **DataPoints** collection is accessed through the **SeriesCollection** object.

Important The DataPoints collection contains only one member at this time. To access it, you must use the 1, as shown below

```
MSChart1.Plot.SeriesCollection(1).DataPoints(-1) _
.Brush.FillColor.Set 0, 255, 255
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

DataPointLabel Object

See Also Example Properties Methods Events

The label for a data point on a chart.

Syntax

DataPointLabel

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

DataPoints Collection

See Also Example Properties Methods Events

A group of chart data points.

Syntax

object.**DataPoints.**(*index*)

The **DataPoints** collection syntax has these parts:

Part	Description
object	An object expression that evaluates to an object the Applies To list.
index	Identifies a specific data point within the current series. For this version of the chart, 1 is the only valid value for this argument. This allows you two make changes to the default settings for all data points in the series. Settings cannot be changed for individual data points within the series.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: DataRepeater Control

Visual Studio 6.0

DataRepeater Control

See Also Example Properties Methods Events

The **DataRepeater** control functions as a scrollable container of data-bound user controls. Each control appears in its own row as a "repeated" control, allowing the user to view several data-bound user controls at once.

Syntax

DataRepeater

Remarks

To use the **DataRepeater** control, you must first create a data-bound user control, and compile it into an .ocx. After creating the user control, the following basic steps must be taken:

1. Add the user control to the project using the **Components** dialog box. This ensures that appropriate files are included by the Visual Basic Package and Deployment Wizard. It also allows access to the control's public properties, events, and methods.
2. On the Properties window, click **RepeatedControlName** and select the user control from the drop down list.
3. Add a data source, such as the ADO Data Control, to the form, and connect it to a data provider.
4. Set the DataRepeater control's **DataSource** property to the data source.
5. Right-click the **DataRepeater** control and click **DataRepeater Properties**.
6. Click the **RepeaterBindings** tab.
7. Set the **PropertyName** to an appropriate **DataField**, and click the **Add** button.

The **DataRepeater** control saves computer resources by only displaying a single user control the active control at a time. The other controls displayed are simple images that do not maintain individual connections to the data source, as would happen if several user controls were contained on a form.

Distribution Note When you create and distribute applications that use the **DataRepeater** control, you should install MSDatRep.ocx in the customer's Microsoft Windows System or System32 subdirectory. The Setup Kit included with Visual Basic provides tools to help you write setup programs that install your applications correctly.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DataReport Object

See Also Example Properties Methods Events

The **DataReport** object is the programmable object that represents the Data Report designer.

Syntax

DataReport

Remarks

The Data Report generates reports using records from a database. To use it:

1. Configure a data source, such as the Microsoft Data Environment, to access a database.
2. Set the **DataSource** property of the **DataReport** object to the data source.
3. Set the **DataMember** property of the **DataReport** object to a data member.
4. Right-click the designer and click **Retrieve Structure**.
5. Add appropriate controls to the appropriate sections.
6. Set the **DataMember** and **DataField** properties for each control.
7. At run time, use the **Show** method to display the Data Report.

Use the **DataReport** object to programmatically change the appearance and behavior of the Data Report by changing the layout of each **Section** object.

The Data Report designer also features the ability to export reports using the **ExportReport** method. This method allows you to specify an **ExportFormat** object, from the **ExportFormats** collection, to use as a template for the report.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: Windows Controls

Visual Studio 6.0

DateTimePicker Control

See Also Example Properties Methods Events

The **DateTimePicker** control enables you to provide a formatted date field that allows easy date selection. In addition, users can select a date from a dropdown calendar interface similar to the **MonthView** control.

Syntax

DTPicker

Remarks

The **DateTimePicker** control, operates in two modes:

- Dropdown Calendar mode (default) enables the user to display a dropdown calendar that can be used to select a date.
- Time Format mode enables the user to select a field in the date display (i.e. the month, day, year, etc.) and press the up/down arrow to the right of the control to set its value.

You can customize the appearance of the drop-down calendar of the control. Various color attributes such as **CalendarBackColor**, **CalendarForeColor**, **CalendarTitleBackColor**, **CalendarTitleForeColor** and **CalendarTrailingForeColor** enable you to create a unique color scheme.

The control can be navigated using either the keyboard or mouse. The drop-down calendar has two buttons that enable you to scroll months in and out of view.

Note The **DateTimePicker** control is part of a group of ActiveX controls that are found in the MSCOMCT2.OCX file. To use the **DateTimePicker** control in your application, you must add the MSCOMCT2.OCX file to the project. When distributing your application, install the MSCOMCT2.OCX file in the user's Microsoft Windows System or System32 directory. For more information on how to add an ActiveX control to a project, see "Adding Controls to a Project" in the *Programmer's Guide*.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: DataCombo/DataList Controls

Visual Studio 6.0

DBCombo Control

See Also Example Properties Methods Events

The **DBCombo** control is a data bound combo box with a drop-down list box which is automatically populated from a field in an attached **Data** control, and optionally updates a field in a related table of another **Data** control. The text box portion of **DBCombo** can be used to edit the selected field.

Syntax

DBCombo

Remarks

The **DBCombo** control differs from the standard **ComboBox** control. While the **ComboBox** control list is filled using the **AddItem** method, the **DBCombo** control is automatically filled with data from a field in the **Recordset** object of a **Data** control to which it is attached. The standard **ComboBox** control must be populated manually by using the **AddItem** method. In addition, the **DBCombo** control has the ability to update a field within a related **Recordset** object which may reside in a different **Data** control.

The **DBCombo** control also supports an automated search mode that can quickly locate items in the list without additional code.

Shown below is a list of the properties used to fill and manage the **DBCombo** control, and bind the selected data to a **Data** control.

Property	Specifies
DataSource	Name of **Data** control that is updated once a selection is made.
DataField	Name of a field that is updated in the **Recordset** specified by the **DataSource** property.
RowSource	Name of **Data** control used as a source of items for the list portion of the control.
ListField	Name of a field in the **Recordset** specified by **RowSource** that is used to fill the drop-down list. **DBCombo** does not support fields of **LongBinary** type for the Listfield property.
BoundColumn	Name of a **Field** in the **Recordset** specified by **RowSource** to be passed back to the **DataField** once a selection is made. **DBCombo** does not support fields of **LongBinary** type for the **BoundColumn**.
BoundText	Text value of **BoundColumn** field. Once a selection is made, this value is passed back to update the **Recordset** object specified by the **DataSource** and **DataField** properties.
Text	Text value of the selected item in the list.

MatchEntry	How the list is searched as the user types in characters at run time.
SelectedItem	The bookmark of the selected item in the Recordset specified by the RowSource property.
VisibleCount	The number of items visible in the list (fully or partially).
VisibleItems	An array of bookmarks with a maximum number of items equal to the **VisibleCount** property.

Users can search the **DBCombo** control by typing a value into the text box portion of the control. Once entered, this value is located in the list and the current list item is set to that item. If the item is not found, the **BoundText** property is set to null.

Note If you do not make the boundary of the control large enough for at least one row of the dropdown list, the list will not appear at run time.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: DataCombo/DataList Controls

Visual Studio 6.0

DBList Control

See Also Example Properties Methods Events

The **DBList** control is a data bound list box which is automatically populated from a field in an attached **Data** control, and optionally updates a field in a related table of another **Data** control.

Syntax

DBList

Remarks

The **DBList** control differs from the standard **ListBox** control. While the ListBox control list is filled using the AddItem method, the DBList control is automatically filled with data from a field in the **Recordset** object of a **Data** control to which it is attached. The standard **ListBox** control must be populated manually by using the **AddItem** method. In addition, the **DBList** control has the ability to update a field within a related **Recordset** object which may reside in a different **Data** control.

The **DBList** control also supports an automated search mode that can quickly locate items in the list without additional code.

Shown below is a list of the properties you use to fill and manage the list, and bind the selected data to a **Data** control.

Property	Specifies
DataSource	Name of **Data** control that is updated once a selection is made.
DataField	Name of a field that is updated in the **Recordset** specified by the **DataSource** property.
RowSource	Name of **Data** control used as a source of items for the list portion of the control.
ListField	Name of a field in the **Recordset** specified by **RowSource** that is used to fill the list. **DBList** does not support fields of **LongBinary** type for the **ListField** property.
BoundColumn	Name of a field in the **Recordset** specified by **RowSource** that is passed back to the **DataField** once a selection is made. **DBList** does not support fields of **LongBinary** type for the **BoundColumn**.
BoundText	Text value of the **BoundColumn** field. Once a selection is made, this value is passed back to update the **Recordset** object specified by the **DataSource** and **DataField** properties.
Text	Text value of the selected item in the list.
MatchEntry	How the list is searched as the user types characters at run time.
SelectedItem	The bookmark of the selected item in the **Recordset** specified by the **RowSource** property.

VisibleCount	The number of items visible in the list (fully or partially).
VisibleItems	An array of bookmarks with a maximum number of items equal to the **VisibleCount** property.

The **DBList** control will automatically highlight an item in the list if the **BoundText** property becomes equal to the value of the field specified by the **DataSource** and **DataField** properties, such as when using a **Data** control to change the current record.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DEAggregate Object

See Also Example Properties Methods Events

Defines an aggregate field within a **DECommand** object.

Syntax

DEAggregate

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DEAggregates Collection

See Also Example Properties Methods Events

A collection whose elements represent each DEAggregate object within the DECommand object.

Note This collection's **Count** property specifies the number of **DEAggregate** objects that exist in the collection.

Syntax

DEAggregates(*index***)**

The *index* placeholder represents an integer with a range from 1 to `DEAggregates.Count`.

Remarks

Use the following code to access the members of the **DEAggregates** collection by their individual names.

```
MyDEExt.DEAggregates("SumOrders").Name
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic for Applications Reference

Visual Studio 6.0

Debug Object

See Also Example Properties Methods Events Specifics

The **Debug** object sends output to the **Immediate** window at run time.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DECommand Object

See Also Example Properties Methods Events

Stores the design-time properties of an ADO Command object.

Syntax

DECommand

Remarks

This object corresponds to the ADO Command object.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DECommands Collection

See Also Example Properties Methods Events

A collection whose elements represent each **DECommand** object within a DataEnvironment object.

Note This collection's **Count** property specifies the number of **DECommand** objects that exist in the collection.

Syntax

DECommands(*index***)**

The *index* placeholder represents an integer with a range from 1 to `DECommands.Count`.

Remarks

This collection is used in the Data Environment Extensibility Object Model and is different from the **Commands** collection used at run time.

Use the following code to access members of the **DECommands** collection by their individual names.

```
DEExtObj.DECommands("Command1").Name
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DEConnection Object

See Also Example Properties Methods Events

Stores the design-time properties of an ADO Connection object.

Syntax

DEConnection

Remarks

This object corresponds to the ADO Connection object.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DEConnections Collection

See Also Example Properties Methods Events

A collection whose elements represent each **DEConnection** object within a DataEnvironment object.

Note This collection's **Count** property specifies the number of **DEConnection** objects in the collection.

Syntax

DEConnections(*index***)**

The *index* placeholder represents an integer with a range from 1 to `DEConnections.Count`.

Remarks

The **DEConnections** collection is used in the Data Environment Extensibility Object Model and is different from the **Connections** collection used at run time.

Use the following code to access members of the **DEConnections** collection by their individual names.

```
DEExtObj.DEConnections("Connection1").Name
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DEExtDesigner Object

See Also Example Properties Methods Events

The **DEExtDesigner** object exposes itself, and its contained collections, through an object model. The DEExtDesigner object is the top-level container object in the Data Environment object model. This object provides a container for a set of related DEConnection and DECommand objects and collections.

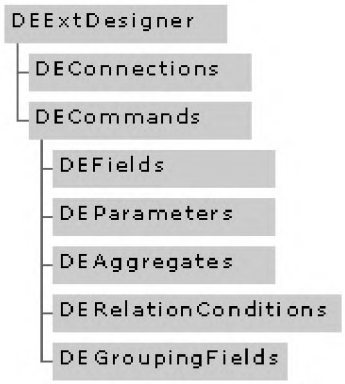
Syntax

DEExtDesigner

Remarks

You can easily define, modify, and use the DEExtDesigner object using the properties, methods, and events associated with each of the objects in the Data Environment object model.

Data Environment Object Model



The Data Environment object model closely matches how the objects are exposed through the user interface. While the Data Environment versions of Connection, Command, Field, and Parameter objects closely match, they are not ActiveX Data Objects (ADO). The objects contained in the Data Environment are used to persist the design-time settings of these objects, and at run time they create true ADO objects, such as Connections, Commands, and Recordsets.

For example, the Data Environment object model does not directly expose the ADO Recordset object. Therefore, the **DEFields** collection in the Data Environment object model branches from DECommands, whereas in the ADO model it branches from the Recordset object.

At design time, you can programmatically set these properties using the Extensibility Object Model. In addition, you can set these properties when designing a DataEnvironment object using the **Command** and **Connection Properties** dialog boxes or Visual Basic's Properties window.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DEField Object

See Also Example Properties Methods Events

Stores the design-time properties of an ADO Field object.

Syntax

DEField

Remarks

This object corresponds to the ADO Field object.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DEFields Collection

See Also Example Properties Methods Events

A collection whose elements represent each DEField object within the DECommand object.

Note This collection's **Count** property specifies the number of **DEField** objects in the collection.

Syntax

DEFields(*index***)**

The *index* placeholder represents an integer with a range from 1 to `DEFields.Count`.

Remarks

The **DEFields** collection is used in the Data Environment Extensibility Object Model and is different from the **Fields** collection used at run time.

Insert the following code to access members of the **DEFields** collection by their individual names.

```
DEExt.DECommands("CustID").DEFields("CustID")
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DEGroupingFields Collection

See Also Example Properties Methods Events

A collection whose elements represent each **DEGroupingField** object within the **DECommand** object.

Note This collection's **Count** property specifies the number of **DEGroupingField** objects in the collection.

Syntax

DEGroupingFields(*index***)**

The *index* placeholder represents an integer with a range from 1 to `DEGroupingFields.Count`.

Remarks

The **DEGroupingFields** collection is used to specify the **DEField** objects by which the **DECommand** is grouped.

Insert the following code to access members of the **DEGroupingFields** collection by their individual names.

```
DEExt.DECommands("CustID").DEGroupingFields("CustID")
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DEParameter Object

See Also Example Properties Methods Events

Stores the design-time properties of an ADO Parameter object.

Syntax

DEParameter

Remarks

This object corresponds to the ADO Parameter object.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DEParameters Collection

See Also Example Properties Methods Events

A collection whose elements represent each **DEParameter** object within the **DECommand** object.

Note This collection's **Count** property specifies the number of **DEParameter** objects in the collection.

Syntax

DEParameters(*index***)**

The *index* placeholder represents an integer with a range from 1 to `DEParameters.Count`.

Remarks

The **DEParameters** collection is used in the Data Environment Extensibility Object Model and is different from the **Parameters** collection used at run time.

Use the following code to access members of the **DEParameters** collection by their individual names.

```
DEExt.DECommands("Param1").DEParameters("Param1").Name
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DERelationCondition Object

See Also Example Properties Methods Events

Defines the relation conditions between the child and parent Command objects of a relation hierarchy.

Syntax

DERelationCondition

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DERelationConditions Collection

See Also Example Properties Methods Events

A collection whose elements represent each **DERelationCondition** object within the **DECommand** object.

Note This collection's **Count** property specifies the number of **DERelationCondition** objects in the collection.

Syntax

DERelationConditions(*index***)**

The placeholder *index* represents an integer with a range from 1 to `DERelationConditions.Count`.

Remarks

Insert the following code to access members of the **DERelationConditions** collection by their individual names.

```
DEExt.DECommands("Relation1")DERelationConditions("Relation1").Name
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: Page Designer

Visual Studio 6.0

DHTMLPage Object

See Also Example Properties Methods Events

Run-time component that hooks up events between the Visual Basic run time and the Dynamic HTML object model.

Syntax

DHTMLPage

Remarks

The **DHTMLPage** object is the run-time component of a Visual Basic **DHTMLPageDesigner** application.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: Page Designer

Visual Studio 6.0

DHTMLPageDesigner Object

See Also Example Properties Methods Events

Represents the design-time page designer. Not available at run time.

Remarks

The **DHTMLPageDesigner** object's properties are only available from within the Visual Basic environment when you build a project using the **DHTMLPageDesigner**. The items under **DHTMLPageDesigner** are design time properties, and as such are not accessible by code.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic for Applications Reference

Visual Studio 6.0

Dictionary Object

See Also Example Properties Methods Events Specifics

Description

Object that stores data key, item pairs.

Syntax

Scripting.Dictionary

Remarks

A **Dictionary** object is the equivalent of a PERL associative array. Items, which can be any form of data, are stored in the array. Each item is associated with a unique key. The key is used to retrieve an individual item and is usually a integer or a string, but can be anything except an array.

The following code illustrates how to create a **Dictionary** object:

```
Dim d                   'Create a variable
Set d = CreateObject(Scripting.Dictionary)
d.Add "a", "Athens"     'Add some keys and items
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
...
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DirListBox Control

See Also Example Properties Methods Events

A **DirListBox** control displays directories and paths at run time. Use this control to display a hierarchical list of directories. You can create dialog boxes that, for example, enable a user to open a file from a list of files in all available directories.

Syntax

DirListBox

Remarks

Set the **List**, **ListCount**, and **ListIndex** properties to enable a user to access items in a list. If you also display the **DriveListBox** and **FileListBox** controls, you can write code to synchronize them with the **DirListBox** control and with each other.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic for Applications Reference

Visual Studio 6.0

Drive Object

See Also Example Properties Methods Events Specifics

Description

Provides access to the properties of a particular disk drive or network share.

Remarks

The following code illustrates the use of the **Drive** object to access drive properties:

```
Sub ShowFreeSpace(drvPath)
    Dim fs, d, s
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set d = fs.GetDrive(fs.GetDriveName(drvPath))
    s = "Drive " & UCase(drvPath) & " - "
    s = s & d.VolumeName  & vbCrLf
    s = s & "Free Space: " & FormatNumber(d.FreeSpace/1024, 0)
    s = s & " Kbytes"
    MsgBox s
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

DriveListBox Control

See Also Example Properties Methods Events

A **DriveListBox** control enables a user to select a valid disk drive at run time. Use this control to display a list of all the valid drives in a user's system. You can create dialog boxes that enable the user to open a file from a list of files on a disk in any available drive.

Syntax

DriveListBox

Remarks

Set the **List**, **ListCount**, and **ListIndex** properties to enable a user to access items in the list. If you also display the **DirListBox** and **FileListBox** controls, you can write code to synchronize them with the **DriveListBox** control and with each other.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic for Applications Reference

Visual Studio 6.0

Drives Collection

See Also Example Properties Methods Events Specifics

Description

Read-only collection of all available drives.

Remarks

Removable-media drives need not have media inserted for them to appear in the **Drives** collection.

The following code illustrates how to get the **Drives** collection and iterate the collection using the **For Each...Next** statement:

```
Sub ShowDriveList
    Dim fs, d, dc, s, n
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set dc = fs.Drives
    For Each d in dc
        s = s & d.DriveLetter & " - "
        If d.DriveType = Remote Then
            n = d.ShareName
        Else
            n = d.VolumeName
        End If
        s = s & n & vbCrLf
    Next
    MsgBox s
End Sub
```

© 2017 Microsoft