

mv.NET Binding Objects



Developer's Introductory Guide

A product from BlueFinity



Copyright Notices

Copyright BlueFinity 2004
Document ref: mvNet_BO_01
Revision 1.0
All rights reserved BlueFinity 2004

Contacting Us

We are always very happy to be able to discuss all aspects of our products with our customers – prospective and current alike. You can contact us via the following means:

Website: www.bluefinity.com
Email: support@bluefinity.com
Telephone: +44 (0) 1442 450 435
Address: Hamilton House
111 Marlowes
Hemel Hempstead
Hertfordshire HP1 1BB
United Kingdom

Trademark Acknowledgements

The mv.NET product and logo are trademarks of BlueFinity International Limited.

All other trademarks and trade names are the property of their respective owners and are used in this documentation for identification purposes only

Contents

mv.NET Binding Objects	1
Copyright Notices.....	2
Contacting Us.....	2
Trademark Acknowledgements.....	2
Welcome to mv.NET	1
The mv.NET Family of Products.....	1
Feature Overview.....	2
The mv.NET Suite.....	2
Getting Started Guide Contents.....	3
Databinding Basics	4
Installation.....	4
The Case for Databinding.....	4
mv.NET's Approach to Databinding.....	5
Binding Objects' Component Set.....	5
An Overview of the Databinding Process.....	6
The PrimaryDataSource Control	7
Summary of Functions.....	7
Using the PrimaryDataSource Control.....	8
The Databinding Designer Window.....	11
The Column Binding and Layout Window.....	14
Lookup Style Editing.....	16
The ILookupTarget Interface.....	19
Linking Multiple PDS Controls.....	21
Batched Updating.....	23
Class Interface Overview.....	24
The SecondaryDataSource Control	31
Why Use a SecondaryDataSource?.....	31

Basic Principles.....	32
Class Interface Overview.....	32
The mvNavigator Control	34
mvNavigator Overview.....	34
Class Interface Overview.....	35
The mvQuery Control	36
mvQuery Overview.....	36
mvQuery Class Interface Overview.....	37
mvQueryList Class Interface Overview.....	38
mvQueryRow Class Interface Overview.....	39
mvQueryColumn Class Interface Overview.....	40

Welcome to mv.NET

Firstly, thank you for either purchasing one or more of the mv.NET products, or for taking the time to explore the great functionality that they can provide to you and your fellow developers.

This chapter outlines the members of the mv.NET family of products and also summarizes the contents of this guide.

The mv.NET Family of Products

Binding Objects is one of the members of the mv.NET family of products authored by BlueFinity. mv.NET is *the* essential tool for any multivalued database developer wishing to create .NET based application interfaces to their current or new multivalued database file system.

The design goal of mv.NET is to enable the multivalued developer to combine the power and flexibility of proven multivalued technology with the start-of-the-art, feature rich .NET environment. Its design also enables and encourages the developer to leverage, wherever possible, previously acquired multivalued skills.

BlueFinity's team of software engineers has huge knowledge and experience of using both multivalued systems and the .NET environment. We proudly regard ourselves as being one of the foremost companies in providing this technology bridge and look forward to working with you to enable you to meet your software development goals.

Feature Overview

The Binding Objects product provides advanced databinding capabilities to the .NET developer.

The Binding Objects architecture has been designed with both performance and flexibility in mind. This, combined with an implementation that provides seamless integration with the .NET environment, provides a powerful tool for enabling mv developers to harness the full power of both their mv system and the .NET platform.

Binding Objects also has strong integration with Microsoft's Visual Studio.NET product, providing improved ease of use and flexibility when compared with Visual Studio's/.NET's native databinding support.

The product's key features are as follows:

- Multivalue aware databinding methodology, allowing full access to all 3 dimensions of item data.
- Multivalued hierarchy support, allowing controls to be connected and interrelated based on multivalued data nesting within a single item.
- Full integration with mv.NET's fetch-on-demand technology.
- Support for pessimistic and optimistic locking.
- Many advanced RAD features to boost developer productivity.

The mv.NET Suite

Binding Objects is one of three products within the mv.NET suite; the suite as a whole comprising of:

- **Core Objects** – object oriented native .NET access to mv databases.

- **Binding Objects** – high performance databinding technology that enables standard .NET controls to become fully mv-aware. Binding Objects links with Core Objects to provide its functionality.
- **Adapter Objects** – complete implementation of an ADO.NET managed data provider for multivalue databases, offering a standardized interface to database access.

Getting Started Guide Contents

The contents of this guide are designed to provide a basis for learning about the Binding Objects module. Further help is provided within the Visual Studio environment using the product's dynamic and intellisense help systems. The chapters of this guide are as follows:

[Databinding Basics](#)

This chapter describes mv.NET's databinding concepts, as well as providing an overview of the primary components that are supplied as part of the Binding Objects module.

[The PrimaryDataSource Control](#)

The PrimaryDataSource control lies at the heart of the Binding Objects product. This chapter describes its workings and explains the different ways in which it may be used to provide advanced databinding functionality.

[Secondary Datasources](#)

Within an application there are typically many places where 'lookup' style information is required. Binding Objects' secondary datasource options allow you to easily defines the source and contents of this lookup information.

Databinding Basics

Databinding, if used correctly, can provide a powerful boost to programmer productivity. This chapter describes the databinding approach incorporated within Binding Objects and provides an overview of the set of tools that come with the product.

Installation

Binding Objects is installed as part of mv.NET's Client Interface Developer setup routine. Please refer to the Getting Started and Core Objects guides for further information on this topic.

The Case for Databinding

The concept of databinding is not new. It essentially revolves around the concept of reducing the programming burden by allowing developers to hook into frequently required data-oriented interface features and capabilities by setting properties of controls at design or (less commonly) run-time – as opposed to writing code.

Databinding has frequently been regarded as a 'soft' option for developers, one which cannot be used for 'serious' application development. This attitude has been historically encouraged by half-hearted implementations of databinding functionality by IDE and software environment vendors.

However, at BlueFinity we take databinding very seriously. We regard it as a key tool in lightening the burden placed upon application developers by today's ever

increasing demand for application sophistication. Implemented properly, it can provide significant savings in terms of development timescales for both the simplest and most advanced business applications.

mv.NET's Approach to Databinding

The Binding Objects module of mv.NET is dedicated to providing databinding technology that pays more than simple lip-service the concept of programming without code. Not only does Binding Objects provide RAD capabilities but it does so in such a way as to allow the full flexibility of the multivalued data model to be embraced within an application.

This is in sharp contrast to the limited, 2-dimensional databinding offered natively within .NET.

To complement the components that come as part of the Binding Objects package a series of Visual Studio add-ins are provided to make the use of the components even more convenient and intuitive.

It is worth noting at this stage that the Binding Objects components leverage extensively the data that can be entered into mv.NET's extended dictionary definitions and so, after reading this guide, you may wish to spend a little time 'fleshing-out' this extended information.

Binding Objects' Component Set

Below are the main components that form the Binding Objects module:

- PrimaryDataSource Control – the main databinding manager component and link to the database.
- SecondaryDataSource Control – provides a secondary data feed to the PrimaryDataSource control, typically used to provide look-up or drop-down option lists.
- mvNavigator Control – provides VCR button type navigation amongst the items of a PrimaryDataSource.
- mvQuery Control – allow queries previously defined within the Data Manager utility to be executed and displayed within an application.

These components are held within the BlueFinity.mvNET.BindingObjects.dll assembly file, which will have been registered in the Global Assembly Cache as part of the CIDSetup installation process. mv.NET's Data Manager add-in for Visual Studio will make them available for use within an mv.NET tab inside VS.NET's Toolbox window.

An Overview of the Databinding Process

In order to incorporate Binding Objects' databinding technology within your application you will need to follow these simple steps:

1. Drop a PrimaryDataSource (PDS) control onto your form. A wizard will then ask you to specify which data file you wish to bind to. The wizard will also allow you to add databound controls at the stage as well.
2. After the PDS has been created, you can use the Properties window to access the databinding definition of the PDS control and any of the controls which are bound to it.
3. You may add extra bound controls by either dropping entries from the Toolbox window and setting the databinding definitions manually, or by using Binding Objects' Control Assistant addin which will automatically set many of the databinding properties for you.
4. For those controls that support the concept of presenting a list of options to the user to select from (e.g. a ComboBox) you can use the Databinding Designer window (accessed via the Properties window) to define which file is to supply the list of options.
5. For input forms that require data from multiple files, you can 'link' multiple PDS controls together to create related parent-child databinding structures. This is done within the databinding design window accessed via the Properties toolwindow.
6. For input scenarios requiring updates to multiple file to be treated as a single atomic update operation, you may use the 'batched updates' feature of the PDS control.

The PrimaryDataSource Control

The PrimaryDataSource control is *the* most important control of the Binding Objects package. It performs a number of key functions and is thus, logically, the first step in learning about the Binding Objects product. This chapter takes you through all of the roles performed by this control and explains how it may be utilized within your application. Within this chapter the acronym 'PDS' is used to denote the PrimaryDataSource control.

Summary of Functions

The PrimaryDataSource control is responsible for providing the following functionality:

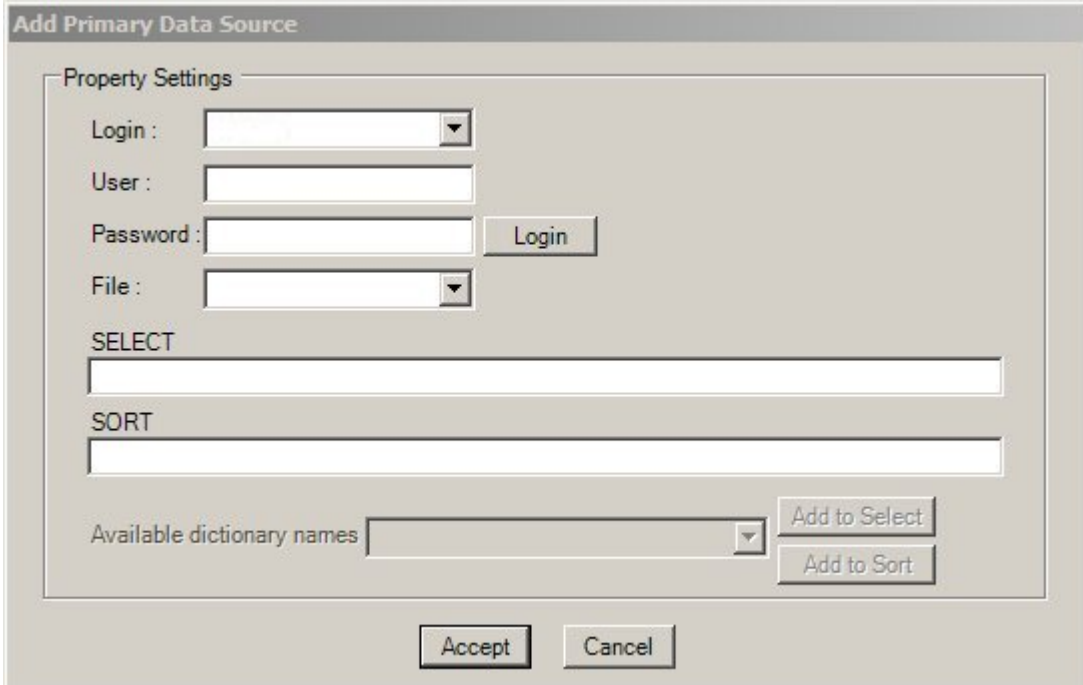
- Primary link to a database file
- Databinding management and event-interception of bound 'child' controls
- Extended property provider for the VS.NET properties window
- Databinding designer form

Many of the above functions can be controlled by both design-time property settings and via run-time programmatic use of its interface members.

Using the PrimaryDataSource Control

The installation of the Client Interface Developer package will have created an mv.NET tab within the Visual Studio Toolbox window. Within this tab is an icon representing the PDS control.

Upon dragging and dropping a PDS control onto the surface of a form, the following pop-up window will be displayed allowing you to specify the database and file to be associated with the new PDS.



The image shows a dialog box titled "Add Primary Data Source". It contains a "Property Settings" section with the following controls: a "Login" dropdown menu, a "User" text box, a "Password" text box with a "Login" button to its right, and a "File" dropdown menu. Below these are two text boxes labeled "SELECT" and "SORT". At the bottom of the dialog, there is an "Available dictionary names" dropdown menu and two buttons: "Add to Select" and "Add to Sort". At the very bottom of the dialog are "Accept" and "Cancel" buttons.

Diagram 3a : PrimaryDataSource Creation Window

Note, the Data Manager addin MUST be running for the above form to appear.

As well as allowing you to specify a login profile and file name, you are also able to set the value of the new PDS's SelectionClause and SortClause properties. Please refer to the [Class Interface Overview](#) section for more details on these properties.

Upon accepting the above form, you will be asked whether you wish to create some databound controls to be associated with the new PDS. If you click 'Yes', the following window will be displayed:

Databound Control Creation

Creation style: Individual controls

Create field prompt labels Append the following string to all prompts: X-location (left) offset: 10 Y-location (top) offset: 10

Please tick the relevent Create checkboxes to indicate which dictionary fields to create

Dictionary Fields	Create	Order	Control Type	Control Name	Prompt Text	Secondary Data Source
NUMBER	<input type="checkbox"/>				Org ID	
NAME	<input type="checkbox"/>				Name	
TYPE	<input type="checkbox"/>				Type	
TYPEDESC	<input type="checkbox"/>				Typedesc	
ADDRESS	<input type="checkbox"/>				Address	
ZIPCODE	<input type="checkbox"/>				Zip code	
ONHOLD	<input type="checkbox"/>				On hold	

Create control(s) within container: Form

Linked Buttons

Create a Save Button Create a Cancel Button Create a New Button Create a Delete Button Create a Navigation Bar

Accept Cancel

Diagram 3b : Databound Control Creation Window

At the top of this form, you may specify how each field name that you select within this form is to be represented. You have 2 basic choices: either create a separate control for each field or create a single grid and have each field represented by a column within the grid.

For individual controls you are able to define whether Label controls are to be created to hold prompt text for the controls. These Label controls re positioned to the immediate left on the created controls. The text within the prompt labels can be defined within the grid in the middle of the form.

If you select a creation style other than *individual controls*, you may specify the data level of the grid as follows:

- Multi-item grid – each row represents one data item
- Multivalued grid – each row represents a multivalue position (for a group of attributes)
- Subvalued grid – each row represents a subvalue position (for a group of attributes)

Also, for a grid, you may specify the grid control type, its name and also whether a row pointer column is to be displayed in the left-most column within the grid.

For all types of control creation you are able to specify the top/left position of the first control via the 2 offset input fields at the top of the form. Multiple individual controls will be created one above another.

In the center of the form you are able to select which fields are to be bound. The following columns are provided so that you may define the creation process at a more detailed level:

Order : For individual controls this column determines the top-to-bottom position. For grid controls this determines the left-to-right position of the associated column.

Control Type : For individual controls this column determines the type of control used to represent the field on the form. For grid controls this determines the type of editor used within the grid column.

Control Name : (Only available for individual controls) This allows the name of the control to be specified.

Prompt Text : (Only available for individual controls) This allows the wording of the prompt text associated with the control to be specified.

SecondaryDataSource : (Only available for combo and lookup style input) This allows the source of options within the control/grid column to be specified.

In addition to selecting the fields to be represented by databound controls, you may also specify the container to hold the new controls. This defaults to the parent form's surface, but you may change this to a new Panel control or a new GroupBox control. For the latter of these 2 options, you are able to specify the name of the new control and for the GroupBox, you are able to bind its Text property to a field. Note, for the GroupBox binding, the value of the specified field will be appended to any design-time content of the Text property.

Finally, the control creation window allows you to request the creation of some maintenance control buttons, which will be associated with the PDS.

Upon clicking the OK button, the requisite controls will be created on the form and the databinding definition information automatically completed.

The Databinding Designer Window

The PDS control, amongst other things, is responsible for adding an extra entry into the VS.NET Properties window for controls that it recognizes as being potential databound children. The name of this extra property takes the form:

'Databinding on pdsxyz'

Where 'pdsxyz' represents the name of the PDS. If you have multiple PDSs on a form, you will get one of these extra Property window entries for each PDS.

If you select one of the 'Databinding on' entries within the Properties window, an ellipsis button will be shown. On clicking this button, the databinding designer window will be displayed for the currently selected control. The contents of the databinding designer form will vary according to what details are selected/specified within the designer form itself. An example is shown below:

The screenshot shows the 'DataBinding Definition' dialog box. It features a list of 'Bound controls' on the left, with 'cbTypedesc' selected. The main area contains several configuration fields: 'Control type' is set to 'CheckedListBox', 'Binding type' is 'SingleItem', and 'Binding level' is 'Attribute'. The 'Bound to field' is 'TYPEDESC'. Below this is a 'Secondary DataSource Definition' section with a dropdown for 'ORGANIZATIONTYPE', 'Load data' set to 'OnPDSLoad', a text box for 'Selection/Sort clause' containing 'SELECT ORGANIZATIONTYPE', a text box for 'Field update control' containing 'TYPE=TYPECODE', and a text box for 'Fields to display' containing 'DESCRIPTION'. At the bottom are 'Accept', 'Cancel', and 'Remove' buttons.

Diagram 3c : DataBinding Definition Window

On the left-hand side of the form is a list of all of the controls bound to the PDS. You may click on any of the entries in this list to view the associated databinding definition, although you will only be able to amend the details for the control currently highlighted within the WinForm designer. Below is a complete list of the possible input fields that may be shown within the databinding designer form:

Binding Type : Allows you to specify whether the control is to be bound to a single or multiple items. The multiple items option here is only available for a subset controls. For Button controls you are also presented with an 'Action' option which should be used when binding a Button control to one of the PDS item maintenance actions (Save, Cancel, Delete and New). A Button control may also be assigned an action type of *Lookup*, in which case it will act as a value lookup trigger when clicked. Please refer to the [Lookup Style Editing](#) section below for more details on this topic.

Binding Level : The sub-item binding level. This may be set at the attribute, multivalue or subvalue level. If *subvalues* is selected, an extra '**MV positioning parent**' prompt will be displayed allowing you to specify the name of the control which will provide the multivalue position at runtime; this may be any control that has a Binding Level setting of *multivalues*.

Bound to Attributes : This field allows you to specify the field or fields to which the control is to be bound. For control types that can only be bound to a single field (e.g. a TextBox control), this input field takes the form of a dropdown ComboBox. For control types that may be bound to multiple fields (e.g. grids), this input field takes the form of a non-editable TextBox showing the current list of bound fields and a ellipsis 'builder' button that, when clicked, displays a pop-up window allowing the fields to be selected. This window is covered in the [Column Mapping and Layout Window](#) section

Action Type (only relevant for Button controls) : If *Action* is chosen as the Binding Type, this fields allows you to select the required action.

Partner (only relevant for Button controls) : If *Lookup* is chosen of the Action type, this field allows you to specify which control is to automatically receive the selected value from the lookup form displayed when the button is clicked.

The lower half of the databinding designer window allows you to specify the source of secondary data for those control types (or Column Bindings) that are able to present a list of options from which the user may select a new value for the bound field.

For control types that may be bound onto multiple fields, a ComboBox is displayed within the databinding designer form allowing you to select which field the secondary datasource definition relates to.

A secondary datasource definition consists of 5 main aspects:

- Datasource
- Secondary data load point
- Secondary data selection and sort criteria
- Update control definition
- Display definition

The datasource definition allows you to specify where data is to be selected from to form the secondary data list. It may be either the name of a file within the parent account or the name of a SecondaryDataSource control – see the chapter dedicated to the [SecondaryDataSource](#) control for more details on this topic.

The secondary data load point definition allows you to control when the data that forms the list of options is to be retrieved from the secondary datasource. The option that you select here will depend entirely upon the context of the application. Below are the options available:

StaticList – secondary data will be assembled the first time that the parent PDS is loaded, the list of options being cached (and not refreshed) throughout the lifetime of the specific application invocation. This option should be used if the option data is fundamentally static in nature.

OnPDSLoad – secondary data will be assembled each time the parent PDS is loaded. It will be cached throughout the lifetime of the specific instance of that PDS. This option should be selected if the option data is reasonably static and is unlikely to change throughout the lifetime of the parent form instance.

OnPDSSelect – secondary data will be assembled each time the Select method of the parent PDS is invoked. It will be cached throughout the lifetime of the item list generated by the PDS.Select execution.

OnDropdown – secondary data will be assembled each time the list of options is displayed. This option should be used when the option data is highly dynamic in nature, typically changing based on the data content of other controls on the form. It is common (though not mandatory) that the `BeforeSDSSelect` event is used to intercept and manually control the selection process when this option is used so that the selection process is able to incorporate current data values from controls or other runtime sources.

The secondary data selection and sort criteria definitions allow you to control which data is retrieved from the server to form the secondary data list along with the sort order.

The update control definition allows you to control how fields within the parent PDS file are updated using the selected secondary datasource item. To enter the update control definition, click the ellipsis (builder) button associated with this definition. The list of fields displayed within the builder form for this definition contains all of the fields within the PDS file from which the value of the bound field is derived. For non-calculated fields, this will just be the name of the bound field itself. For each field listed from the PDS file, you are able to specify the name of a field from the secondary datasource, the value of which will be used to update the corresponding field within the PDS file when an item from the secondary datasource is selected.

The display definition for a secondary datasource allows you to define which fields from the secondary datasource file are to be displayed to the user. The builder form for this definition is the Column Binding and Layout designer, covered in the [following section](#). This allows you to control the columnar display of option data.

The Column Binding and Layout Window

When the databinding designer window is shown for a control type that may be bound to multiple fields, the Bound to Attributes input field takes the form of a non-editable TextBox showing the current list of bound fields and a button that, when clicked, displays a pop-up window allowing the relevant fields to be selected. The contents of this window are shown below:

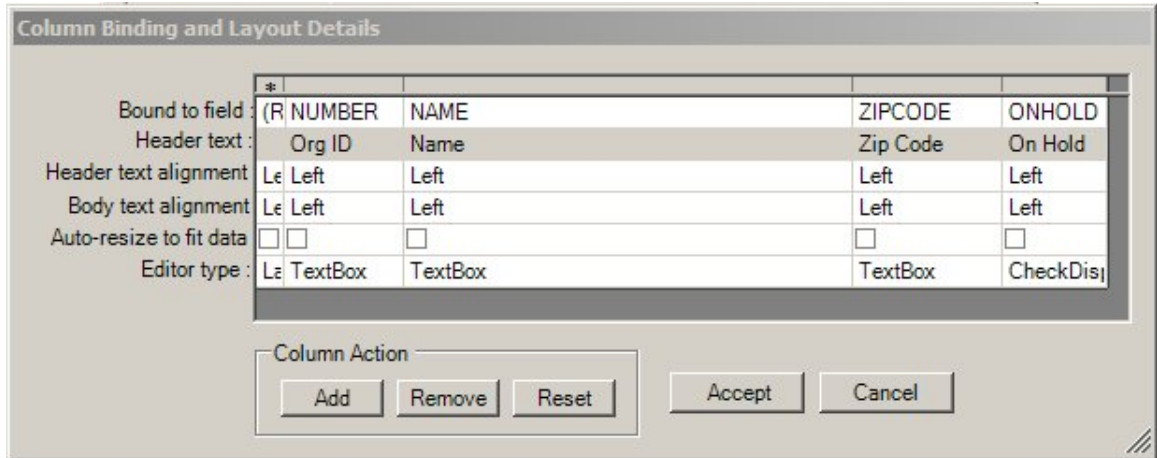


Diagram 3d : Column Binding and Layout Definition Window

All controls that may be bound to multiple fields will present their data content in a columnar style. Hence, the above window allows you to define the content of each column. You may add extra field bindings (columns) by clicking the Add button, or remove existing ones by clicking the Remove button. The Remove button will remove the currently selected column – indicated by the presence of an asterisk in the top header row.

The various rows of the window's grid allow you to define a particular aspect of each column. Note, the 'Auto-resize to fit' data setting may not be supported by all controls. The 'Editor type' setting may be one of the following:

Label (non editable display)

TextBox *

ComboBoxList †

ComboBoxEdit *†

CheckBox *

CheckBoxDisplay

DatePicker *

Lookup †

LookupEdit *†

* only available for non-calculated fields.

† use the secondary datasource capability of mv.NET databinding to assemble the list of options. Secondary datasources are covered in the [Databinding Designer](#) section

The Lookup editor styles force a pop-up form to be displayed at runtime allowing the user to select a value. This topic is explored in the [Lookup Style Editing](#) section below.

Lookup Style Editing

For Grid control columns with a 'Lookup' editor style or for Button controls that are flagged with a 'Lookup' action type, mv.NET provides an easy to use, flexible and customizable value lookup architecture. Lookups are used when a more sophisticated value selection process is required, typically, when:

- there are too many options to list within a combo box
- multiple fields are required to correctly identify options
- a data searching capability is required

- a stylized selection interface is desirable.

At runtime, for grids, an ellipsis button will be displayed at the right edge of the input cell for all columns that have an editor type of 'Lookup'.

Button controls which have a databinding action type of 'Lookup' will be typically positioned at the right-hand edge of a Textbox or Label control (termed the 'partner' control) which will be used to display the selected lookup value. The databinding designer form allows you to identify the 'partner' control of a lookup button if one is being used.

The Databinding Designer window also allows you to specify a secondary datasource to be associated with the lookup action. It is the contents of this secondary datasource that will be shown within a form that is presented when the lookup is requested (i.e. when the lookup button is clicked) by the user.

There are 2 basic options available in terms of what gets displayed when the user requests a lookup – the default lookup form or your own lookup form.

The default lookup form is shown in the screenshot below:

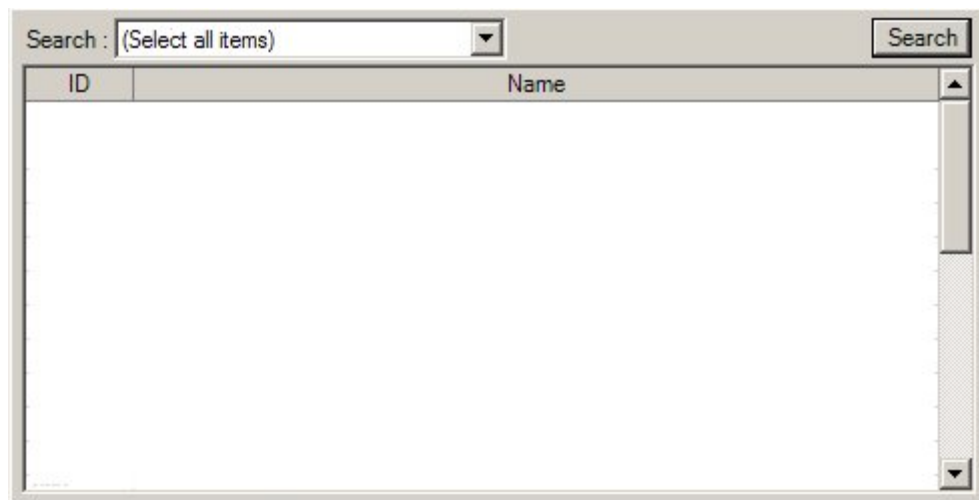


Diagram 3e : The Default Lookup Form

The default lookup form will initially display the first dozen or so items within the secondary datasource and allow the user to scroll through and view other items. The top combobox allows the user to select a dictionary field and, on doing so, 2 further input fields are displayed; one to allow them to choose a selection operator (equal to, less than, greater than, etc) and another to enter a selection value. After entering values into these 2 fields, the user may then click the Search button to initiate a fresh selection of the secondary datasource.

In order to allow the developer to invoke their own form when the user clicks a lookup button the developer must follow 2 main steps:

- a) Create a component (typically a WinForm) and, within this, implement BindingObjects' ILookupTarget interface. This interface requires you to implement 4 or 5 simple methods in order to allow BindingObjects to position and control your form correctly. The members of this interface are documented in the following section.
- b) Instantiate an instance of the custom lookup form within the BeforeLookup event of the parent PDS. The signature of the BeforeLookup event is as follows:

```
Public Event BeforeLookup(ByVal Sender As PrimaryDataSource,  
                          ByVal LookupParent As Control,  
                          ByVal AttributeName As String,  
                          ByVal RowIndex As Int32,  
                          ByVal ColumnIndex As Int32,  
                          ByRef LookupTarget As ILookupTarget,  
                          ByRef DataList As mvItemList)
```

The arguments for this event are:

Sender : A reference to the parent PDS.

LookupParent : A reference to the control which has triggered the lookup action.

AttributeName : The attribute to which the LookupParent control (or LookupParent column if it is a grid) is bound.

RowIndex : If the LookupParent is a grid control, this argument holds the current row position of the grid.

ColumnIndex : If the LookupParent is a grid control, this argument holds the current column position of the grid.

LookupTarget : If you wish to show your own form, you should assign a reference to your instantiated form to this argument.

DataList : If you wish to assemble your own data list to be used as the secondary datasource within the lookup form you should assign a reference to the list to this argument.

Thus, in summary, this event allows you to:

- a) Determine the context in which the lookup action is being invoked.
- b) Instantiate your own lookup form.
- c) Assemble your own data list.

There is also an AfterLookup event which allows you to intercept the value which is being passed back as a result of the user selecting a lookup value. The signature of this event is as follows:

```
Public Event AfterLookup(ByVal Sender As PrimaryDataSource,  
                        ByVal LookupParent As Control,  
                        ByVal AttributeName As String,  
                        ByVal RowIndex As Int32,  
                        ByVal ColumnIndex As Int32,  
                        ByRef Value As Object)
```

The arguments for this event are:

Sender : A reference to the parent PDS.

LookupParent : A reference to the control which has triggered the lookup action.

AttributeName : The attribute to which the LookupParent control (or LookupParent column if it is a grid) is bound.

RowIndex : If the LookupParent is a grid control, this argument holds the current row position of the grid.

ColumnIndex : If the LookupParent is a grid control, this argument holds the current column position of the grid.

Value : The value to be entered into the source control/grid cell. You may amend this value as necessary.

The ILookupTarget Interface

If you wish to have your own form or component displayed as a result of a user clicking a lookup button, you will need to implement BindingObjects' ILookupTarget interface within the component that you create.

The purpose of ILookupTarget is to provide BindingObjects with a standard set of interface members by which it may position and control your form at runtime. The members of the ILookupTarget interface are as follows:

```
Sub InitializeData(ByVal ColumnBindingDefinition As ColumnBindings,  
                 ByVal DataList As Object)  
ReadOnly Property Dimensions() As BlueFinity.mvNET.BindingObjects.Size  
Sub InitializeDisplay(ByVal InitialPosition As ScreenPosition,  
                    ByVal InitialValue As String, ByVal LookupParent As ILookupParent)  
Sub GrabFocus()  
Sub CloseDown()
```

InitializeData

This method allows you to receive the column binding definition for the column/control which is requesting the lookup, along with the data to be displayed within the lookup form. This method is called by BindingObjects just before your component is displayed.

Dimensions

This property allows you to pass back the dimensions of your lookup component. This property is accessed just before the InitializeDisplay method is invoked. An example of the implementation code would be as follows:

```
Return New BlueFinity.mvNET.BindingObjects.Size(Me.Height, Me.Width)
```

InitializeDisplay

This method allows you to position your component and make it visible. Typically, for a WinForm component, you would implement this method with the following code:

```
Me.Top = InitialPosition.Top  
Me.Left = InitialPosition.Left  
Me.LookupParent = LookupParent  
Me.Show()
```

Note, in the above code, LookupParent is a class level variable (which you will need to declare) that allows you to persist the passed reference to the calling component (which will typically be a PrimaryDataSource control). You will need to use this LookupParent reference within your component code later on – see below for more details on this.

GrabFocus

This method is invoked when the input focus needs to be grabbed by your component. Typically your implementation code would be as follows:

```
Me.Activate()
```

CloseDown

This method is invoked when BindingObjects require you to terminate your component. Typically your implementation code would be as follows:

```
Me.Close()
```

As well as implementing the members of the ILookupTarget interface, there are a few places within your lookup component code where you will need to utilize some members of the LookupParent reference passed via the ILookupTarget.InitializeDisplay

method. This LookupParent object will implement the ILookupParent interface. It is the members of this interface that you will need to use – these are as follows:

```
Sub LookupCancelled(ByVal Sender As ILookupTarget)  
Sub ReturnValue(ByVal Sender As ILookupTarget, Value As Object)  
ReadOnly Property GivenFocus() As Boolean
```

LookupParent.LookupCancelled

You need to invoke this method when you detect that the user has requested to cancel the lookup action. Typically this will be used within the Click event handler of a cancel or close button and within the Deactivate event of your component. You need to pass a reference to your component in this method call.

LookupParent.ReturnValue

This method is used to pass back the lookup value selected by the end-user. It should be invoked at the point when the user selects the value. It can be assigned either a string value or an mvltem instance. You also need to pass a reference to your lookup component in this method call.

LookupParent.GivenFocus

This property should be accessed within the Deactivate event handler of your component. Typically you would have the following code in this event handler:

```
If LookupParent.GivenFocus Then  
    LookupParent.LookupCancelled(Me)  
End If
```

Linking Multiple PDS Controls

If you have a requirement to access data from more than one file within a form you will need to use multiple PDS controls on that form. If the data from these files is related in a parent/child fashion, i.e. if when the selected record in the parent file changes a reselection of related items in the child file is required, then you may 'link' PDS controls together in order to force this 'cascaded' style of selecting to happen automatically.

The linking definition for a PDS is entered by clicking the *Linked PrimaryDataSources* ellipsis button at the foot of the databinding designer form accessed via the Property toolwindow's *DataBindings* entry for the 'parent' PDS control. The resulting window is shown below:

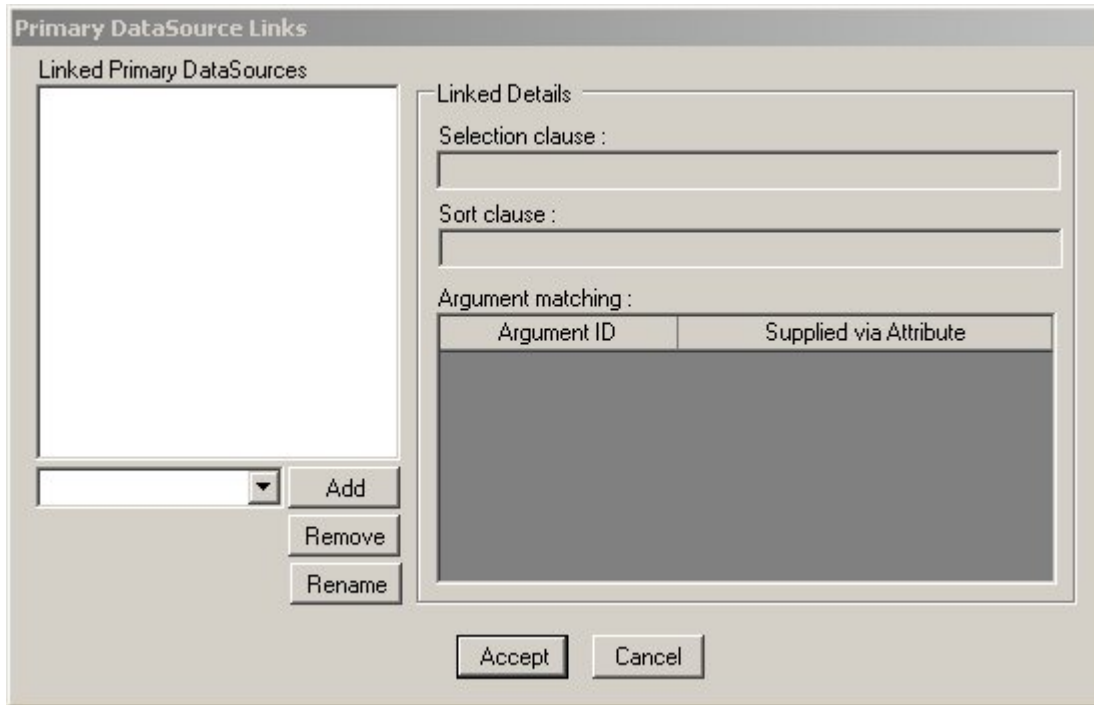


Diagram 3f : Primary DataSource Links Window

The combobox in the mid-left of the *Links* window contains the names of all the PDSs within the parent form. By selecting a PDS name (that represents the PDS which you wish to link to) in the combobox (the PDS that you select from the combobox being the child of the linking relationship) you may then click the *Add* button in order to add it to the *Linked Primary DataSources* list.

By clicking an entry in the *Linked Primary DataSources* list, you will be able to edit/view the linking definition for this particular PDS within the *Linked Details* area to the right of the form. The PDS selected within the *Linked Primary DataSources* list is termed the 'child PDS'.

Within the *Linked Details* area the *Selection clause* displays the value of 'child PDS' *SelectClause* property; likewise the *Sort clause* displays the value of its *SortClause* property. These 2 fields are displayed for reference purposes only.

Within the *Argument matching* grid you are able to specify which fields from the Parent PDS' file are to supply values to the runtime arguments within the child PDS' select and sort clauses - this being the reason why the settings of these 2 properties from the child PDS are shown within the *Linked Details* area. Runtime arguments within the *SelectClause* and *SortClause* properties should be entered as words within curly braces. For example, the following *SelectClause*:

```
TYPE = "{Type}" AND WITH NAME = "{Name}]"
```

will result in the *Argument matching* grid containing 2 rows, one containing the word 'Type' in the Argument ID column, and the other containing the word 'Name' in the same column. Clearly, these 2 rows represent the 2 runtime arguments inserted within the SelectClause. You are then able to select the relevant dictionary name for each row within the 'Supplied via Attribute' column, thus indicating which fields from the Parent PDS are to supply argument values at runtime.

The net result of doing this will be to enable the parent PDS to insert values from its currently selected item into the select and sort clauses of the child PDS prior to item selection occurring - which will then allow the relevant list of items to be selected by the child PDS. This operation will be performed automatically by the parent PDS each time its current selection position (CursorPos property) changes, thus creating the desired cascading selection effect.

PDSs may be linked to any level of nesting.

Batched Updating

There are occasions within an application where a form displays data from multiple files but where updates to these files need to be treated as a single atomic operation. It is for such situations that the PDS control provides its 'batched updating' feature whereby updates to multiple files and possibly multiple items are batched up at the client end and committed to the database in a single operation.

Each PDS control has a *BatchUpdates* property which may be set to one of the following values:

- None - the PDS is not participating in batched updating
- Controller - the PDS is acting as a batched update controller
- Child - the PDS is participating in batched updating and is being controlled by another PDS

A batched update controller is responsible for coordinating the overall update commit/cancel boundary for all of the updates performed on its own items and also for any item updates made on items from other PDSs with a *BatchUpdates* property setting of *Child*. It is similar to a transaction boundary in some respects but has the following important differences:

- it doesn't require transaction boundary support on the database server
- the 'transaction boundary' only covers items within the PDS batch controller's current selection list and any within its child PDSs.

Batched updating works in tandem with PDS linking ([see previous section](#)) in that a batch controller coordinates updates for its child linked PDS controls. When a PDS with a *BatchUpdates* property set to *Child* gets loaded at runtime, the first thing it does is to scan back up the PDS link chain to find its parent batched update controller. This scanning process (and hence batched updating) is limited to 4 levels of link nesting deep.

You may have more than one batched update controller on a form, each one coordinating updates within its own set of child PDSs.

In order to commit/cancel an update batch, the UpdateSave or UpdateCancel method of the batch controller should be invoked, or the BatchSave or BatchCancel methods of any child PDS. Therefore, if you are using bound action buttons for save and cancel actions, these should be bound to the batch controller PDS.

Class Interface Overview

The PrimaryDataSource control has a comprehensive class interface that is accessible both at design time (via the VS.NET properties tool window) and also programatically. The table below gives an overview of each member of the PDS interface. More detailed documentation can be found in the on-line help integrated within Visual Studio.

Name	Description
Account	Property : Reference to the mvAccount instance which is being used/to be used by the PDS control.
AccountName	Property : The name of the account profile to be used by the PDS. This property is read-only if the LoginName property is not null.
Add	Method : Allows a new item to be added to the list held by the PDS control. There is an override for this method which allows a grid control reference to be passed in order to allow extra rows to be inserted within multivalued and subvalue grids.

AfterActionClick	Event : Raised after the work triggered by the click of any one of the action buttons associated with the PDS (Save, Cancel, Delete, New) has been completed.
AfterBatchCancel	Event : Raised after a batched update action has been cancelled.
AfterBatchSave	Event : Raised after a batched update action has been successfully saved.
AfterBatchStart	Event : Raised after a batched update action has been initiated.
AfterLookup	Event : Raised after a value lookup action has been completed.
AfterUpdateCancel	Event : Raised after an item update has been cancelled.
AfterUpdateSave	Event : Raised after an item update has been successfully saved.
AfterUpdateStart	Event : Raised after an item update has been initiated.
AutoGridSort	Property : Indicates whether grids which are bound to this PDS are to support sorting via the clicking of column titles.
AutoGridSortThreshold	Property : Indicates the threshold beyond which grid sorting will be done via a sorted reselection of items by the server as opposed to a complete retrieval of items to the client in order for the sort action to be performed locally.
AutoWrite	Property : Indicates whether items will be automatically written to the database when the PDS's cursor position moves away from a modified item. This property also indicates whether an item should be deleted from the database if the PDS's Delete method is invoked.
BatchCancel	Method : Cancel the batched update action that is currently in progress.
BatchedUpdates	Property : Indicates the style of batched updating to be operated by the PDS.
BatchInProgress	Property : Indicates whether a batched update action is currently in progress.
BatchSave	Method : Save all of the updates performed during the current batched update action to the database.
BeforeActionClick	Event : Raised immediately before the work triggered by the click of any one of the action buttons associated with

	the PDS (Save, Cancel, Delete, New) has been initiated.
BeforeControlRecalc	Event : Raised before the value of a bound control is about to be recalculated.
BeforeLookup	Event : Raised before a value lookup action is performed. This event allows you to set the lookup form and data list is required.
BeforeSDSSelect	Event : Raised before the secondary data is to be assembled. This event allows you to supply customized secondary data.
BeforeUpdateCancel	Event : Raised before an item update is about to be cancelled.
BeforeUpdateSave	Event : Raised before an item update is about to be saved.
BoundChild	(Read-only) Property : Returns a reference to a single databinding within the PDS.
BoundChildren	(Read-only) Property : Returns all of the databindings within the PDS.
ButtonCancel	Property : The name of the Button control which will be automatically used as the cancel unsaved amendments button.
ButtonDelete	Property : The name of the Button control which will be automatically used as the delete current item button.
ButtonNew	Property : The name of the Button control which will be automatically used as the create new item button.
ButtonSave	Property : The name of the Button control which will be automatically used as the save amendments button.
Clear	Method : Resets the PDS's internal ItemList property to an empty list and clears the contents of all bound controls.
ControlsLocked	Method : Allows all bound controls to be locked/unlocked in unison.
CurrentItem	(Read-only) Property : Returns a reference to the mvItem (within the PDS's ItemList) indicated by the current value of the CursorPos property.
CursorMoved	Event : Raised when the CursorPos property changes.
CursorPos	Property : The current/desired value of the ItemList's CursorPos property. Changing this property will cause all bound controls to be repopulated with the data of the item at the new list position.
Delete	Method : Deletes the item at the current cursor position.

	There is an override for this method which allows a grid control reference to be passed in order to allow rows to be deleted from multivalue and subvalue grids.
DeleteConfirmMsg	Property : The text of the message to be automatically displayed to allow item delete confirmation.
Display	Method : Allows a reference to an mvItem instance to be passed to the PDS for display.
DisplayValidationErrors	Property : Indicates whether a popup message window containing an error description is to be automatically displayed upon encountering a validation error.
EnterAddMode	Method : Forces the PDS to enter into its new item creation mode. This method is useful when a new item has been manually added to the PDS's ItemList prior to the PDS being loaded.
EnterKeyHandling	Property : Indicates how the Enter key is to be handled in terms of control navigation.
ErrorStack	(Read-only) Property : Returns an array of strings holding any errors that have been encountered during the most recent PDS action.
ErrorStackText	(Read-only) Property : Returns the data in the ErrorStack property as a printable string.
ExclusiveConnection	Property : Indicates whether this PrimaryDataSource needs an exclusive (not shared with other PDSs) database connection.
File	Property : Reference to the mvFile instance which is being used/to be used by the PDS control.
FileName	Property : The name of the file to be accessed by the PDS.
FileOpen	Method : Manually forces the file specified by the FileName property to be opened.
GridSorted	Event : Raised after a grid bound to the PDS as been sorted.
ID	(Read-only) Property : The item ID of the current item within the PDS's ItemList.
IDList	(Read-only) Property : Returns an array of strings represented all of the item IDs within the PDS's ItemList.
InternalError	Event : Raised when an internal error within the PDS is encountered.
Item	(Read-only) Property : Allows a reference to any of the

	items within the PDS's ItemList to be obtained.
ItemList	Property : The list of items currently selected (and being displayed) by the PDS.
ItemModified	(Read-only) Property : Indicates whether the current item (being displayed) has been modified.
ItemModifiedMsg	Property : The text of the message to be automatically displayed if a form is closed with unsaved modifications present.
LockingError	Event : Raised when a lock has failed to be taken on the current item.
LockingStyle	Property : Indicates the style of locking to be used by the PDS.
Login	Method : Manually forces the PDS control to establish a database connection to the server/account specified by the LoginName property.
LoginName	Property : The name of the login profile to be used to control database connection.
Logout	Method : Forces the PDS to disconnect from the database.
MoveNext	Method : Forces the current cursor position to be advanced by one position.
MovePrev	Method : Forces the current cursor position to be moved back by one position.
Navigator	Property : The name of the mvNavigator control to be bound to the PDS.
PartialItemReads	Property : Indicates whether the whole item string is to be read and returned to the client as opposed to only the explicitly referenced attributes.
Password	Property : The Password to be used in connection action.
Read	Method : Allows an explicit item ID to be read and displayed by the PDS. The current ItemList is lost by the use of this method.
ReBind	Method : Allows the databinding definition of a child control to be reset. This method should be used when the column binding definition of a control has been altered programatically.
RefreshData	Method : Allows the data displayed by a bound control to be redisplayed using the current item's value. This method should be used when the value of attribute within

	the current item has been changed programmatically.
RequestNewID	Event : This event is raised when a new item is about to be saved. It allows you to both enter values into the current item prior to the save to database being performed and also to supply the required item ID if necessary.
Select	Method : Allows the PDS's select action to be invoked programmatically.
SelectClause	Property : The selection clause to be used by the Select method. For example, WITH NAME = "E"]".
SelectControl	Property : The mvSelect control which has been used/ which is to be used during the PDS's select action.
SelectDemandSize	Property : The number of items to be retrieved upon each subsequent server-round trip after initial item selection.
SelectInitSize	Property : The number of items to be retrieved immediately after items have been initially selected.
SelectOnLoad	Property : Indicates whether an item selection action will be automatically performed upon component load.
SelectOnLogin	Property : Indicates whether an item selection action will be automatically performed upon establishing an account login session.
SelectStyle	Property : The style of data retrieval after items have been initially selected.
ServerName	Property : The name of the server profile to be used by the PDS. This property is read-only if the LoginName property is not null.
SortClause	Property : The sort clause to be used by the Select method. For example, BY NAME.
UpdateCancel	Method : Allows the current item modify action to be cancelled.
UpdateCancelFailed	Event : Raised when the UpdateCancel method has failed.
UpdateSave	Method : Allows the current item modify action to be saved to the database.
UpdateSaveFailed	Event : Raised when the UpdateSave method has failed.
UserName	Property : The Name of the user to be used in the PDS's connection action.
Validated	Event : Raised when a control has passed data validation.
ValidationError	Event : Raised when a control has failed to pass data

	validation.
--	-------------

The SecondaryDataSource Control

This chapter cover the purpose and use of Binding Objects' SecondaryDataSource control and explains how it may be utilized within your application. Within this chapter the acronym 'SDS' is used to denote the SecondaryDataSource control.

Why Use a SecondaryDataSource?

Within Binding Objects there are numerous places where you are able to specify whereabouts option data for a control or grid column is to be drawn from – this is termed a secondary data source. In all of the places where you may define secondary data, you may either specify the name of a file or the name SecondaryDataSource control as the source of such data. The main advantages of using an SDS control are:

- a) You are able to control the fetch-on-demand parameters used when the secondary data is assembled.
- b) You are able to reuse or share a single secondary data source amongst a group of controls.
- c) Option data needs to be drawn from a different database.

The main disadvantage of using an SDS control is simply that you need to explicitly create it before you can use it. The simple guideline is, therefore: only use an SDS if

you need to utilize one of the above 3 capabilities, otherwise, specify secondary data sources via simple file name.

Basic Principles

The use of an SDS control to obtain option data is a 2-step affair. Firstly, you create an SDS instance – an SDS icon is placed within the VS.NET Property window's mv.NET tab. After creating an SDS instance, you may set the appropriate entries within the Property window's grid.

Secondly, within the databinding definition for a control (accessed via the 'Databinding on' extended property for the control) you may set the selection criteria to be used for that specific control.

At runtime, the 2 sets of definitions are combined together in order to obtain the relevant option data. The SDS control has no visible interface.

Class Interface Overview

The SecondaryDataSource control's class interface is accessible both at design time (via the VS.NET properties tool window) and also programatically. The table below gives an overview of each member of the SDS interface. More detailed documentation can be found in the on-line help integrated within Visual Studio.

Name	Description
Account	Property : Reference to the mvAccount instance which is being used/to be used by the SDS control.
AccountName	Property : The name of the account profile to be used by the SDS. This property is read-only if the LoginName property is not null.
File	Property : Reference to the mvFile instance which is being used/to be used by the SDS control.
FileName	Property : The name of the file to be accessed by the SDS.
ItemList	Property : The list of items currently selected (and being displayed) by the SDS.
SelectControl	Property : The mvSelect control which has been used/ which is to be used during the SDS's select action.

SelectDemandSize	Property : The name of the server profile to be used by the SDS. This property is read-only if the LoginName property is not null.
SelectInitSize	Method : Allows a new item to be added to the list held by the SDS control. There is an override for this method which allows a grid control reference to be passed in order to allow extra rows to be inserted within multivalue and subvalue grids.
SelectStyle	Property : The style of data retrieval after items have been initially selected.
ServerName	Property : The name of the server profile to be used by the SDS. This property is read-only if the LoginName property is not null.
UserName	Property : The Name of the user to be used in the SDS's connection action.

The mvNavigator Control

This chapter covers the use of Binding Objects' mvNavigator control.

mvNavigator Overview

The mvNavigator control allows a convenient and familiar method of providing the end-user with the means of controlling the current cursor position within a set of selected items associated within a PrimaryDataSource control.

An entry for the mvNavigator control is placed within the VS.NET Property window's mv.NET tab.

On dropping an mvNavigator control onto a form, a VCR-like series of buttons is displayed within the control. At runtime, the user is able to use the buttons within the mvNavigator control to adjust the current cursor position within the associated PDS.

In order to associate an mvNavigator control within a PDS you must set the Navigator property of the PDS to the name of the relevant mvNavigator control.

Class Interface Overview

The mvNavigator control's class interface is very simple, and beyond the standard control position and size type standard properties there are only 4 events that warrant documentation. These are detailed in the table below:

Name	Description
MoveFirst	Event : Raised when the move to first item button of the control is clicked.
MoveLast	Event : Raised when the move to last item button of the control is clicked.
MoveNext	Event : Raised when the move to next item button of the control is clicked.
MovePrev	Event : Raised when the move to previous item button of the control is clicked.

The mvQuery Control

This chapter documents the use and functionality of Binding Objects' mvQuery control, which may be used to display the results of previously defined queries.

mvQuery Overview

Within mv.NET's Data Manager utility you are able to define queries. These queries may be run within the Data Manager but if you wish to integrate the query within your rich-client application you may use the mvQuery control as a means of displaying query results within a WinForm component.

At design time all that you need to do is drop an mvQuery control onto a form and set the appropriate properties.

At runtime you then simply need to invoke the Run method of the control in order to trigger the execution and display of a specific query.

The mvQuery control supports the concept of dynamic detail hiding. This manifests itself as a series of checkboxes at the top of the control representing each break level defined within the query. By clicking these checkboxes, the user is able to control the level of detail displayed within the control.

mvQuery Class Interface Overview

The table below summarizes the mvQuery control's class interface. More detailed documentation can be found in the on-line help integrated within Visual Studio. Some of the members of this interface reference sub-classes within the mvQuery control - the interfaces of these sub-classes are documented in subsequent sections within this chapter.

Name	Description
Account	Property : Returns a reference to the mvAccount instance being used by the control. This reference will have been previously passed into the control via its Run method. See below.
AllowDetailHiding	Property : Indicates whether check boxes at the top of the control should be displayed to allow the user to hide detail lines within the query.
ArgumentWindowPrompt	Property : The text of the prompt in the pop-up window used to collect query argument values at runtime.
ArgumentWindowTitle	Property : The title of the pop-up window used to collect query argument values at runtime.
CancelButtonText	Property : The caption text for the Cancel button when argument values are to be captured by the control at runtime.
Clear	Method : Removes all query data from the control.
Click	Event : Raised when a query line within the control has been clicked. This event passes arguments which allow you to identify which row has been clicked.
DoubleClick	Event : Raised when a query line within the control has been double-clicked. This event passes arguments which allow you to identify which row has been double-clicked.
ExecutingText	Property : The text to be displayed whilst a query is being executed.
GrandTotalText	Property : The text to appear alongside the detail hiding checkbox for the grand total break point.
HideDetailMsg	Property : The message to appear alongside any detail hiding check boxes."

QueryList	Property : the mvQueryList instance being used internally by the control. See mvQueryList section below for further details on this class.
Row	Property : Allows a reference to a specific mvQueryRow instance within the control to be obtained.
Run	Method : Allows a database source and query name to be specified in order to trigger the execution and display of a specific query within the control.
RunButtonText	Property : The caption text for the Run button when argument values are to be captured by the control at runtime.
RunError	Event : Raised when an error is encountered whilst executing the Run method.
SelectedIndexChanged	Method : Raised when the currently selected row within the control changes.
SelectedRow	Property : Returns a reference to an mvQueryRow instance representing the currently selected row within the control's results grid.
ShowGridLines	Property : Indicates whether the results grid within the control is to display lines separating lines and columns.
ValidateArguments	Event : Raised after runtime arguments have been entered by the user so that validation may be performed.

mvQueryList Class Interface Overview

An mvQueryList class instance represents the data being displayed within an mvQuery control. Its interface members are summarized in the following table:

Name	Description
Column	Property : Allows a reference to a specific mvQueryColumn instance within the control to be obtained. See mvQueryColumn section below for further details on this class.
Columns	Property : Returns a reference to an mvQueryColumns instance which represents the collection of all of the mvQueryColumn definitions within the query list.

Count	Property : Returns the number of rows within the query list. This will depend on the HideLevel property setting.
ExpandLevel	Property : Returns the current multi/subvalue expansion level within the mvQuery control. This will be one of: 0 = No expansion 1 = All multivalues expanded 2 = All subvalues expanded Note, this property will be adjusted by the parent mvQuery control if the user double-clicks in the left most column header cell within the results grid.
ExpandLevelChanged	Event : Raised whenever the value of the ExpandLevel property changes.
File	Property : Returns an mvFile instance representing the file from which data has been extracted for the query.
HideLevel	Property : Indicates the current level of detail hiding within the query list.
HideLevelChanged	Event : Raised whenever the value of the HideLevel property changes.
QueryDefinition	Property : Returns an mvItem instance representing the query definition being used.
Row	Property : Allows a reference to a specific mvQueryRow instance within the query list to be obtained.
SelectedIndex	Property : Returns the index number of the currently selected row within the query list.
SelectedIndexChanged	Event : Raised whenever the value of the SelectedIndex property changes.
SelectedRow	Property : Returns a reference to an mvQueryRow instance representing the currently selected row within the query list.

mvQueryRow Class Interface Overview

An mvQueryRow class instance represents a single data row within the mvQueryList instance. Its interface members are summarized in the following table:

Name	Description
BreakLevel	Property : Returns the break level represented by the data row. A value of 0 represents a base data row. A value greater than 0 represents a break total/calculation row.
Value	Property : Returns the value of a specified column within the data row.
getValue	Method : Method version of the Value property. This is supplied for C# developers as C# is not able to supply arguments to property extraction members.

mvQueryColumn Class Interface Overview

An mvQueryColumn class instance represents a single column definition within the mvQueryList instance. Its interface members are summarized in the following table:

Name	Description
Alignment	Property : The alignment setting of the column.
AlignmentChanged	Event : Raised whenever the value of the Alignment property changes.
Index	Property : The ordinal position of the column.
Title	Property : The text of the column heading.
TitleChanged	Event : Raised whenever the value of the Title property changes.
Width	Property : The column's width (in pixels).
WidthChanged	Event : Raised whenever the value of the Width property changes.