mv.NET Core Objects

Developer's Introductory Guide

A product from BlueFinity

Copyright Notices

Contacting Us

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

Website: www.bluefinity.com
Email: info@ bluefinity.com
Address: 10260 SW Greenburg Road, Suite 700, Portland, OR 97223, USA
Address: Hamilton House, 111 Marlowes, Hemel Hempstead, Herts, HP1 1BB, UK

Trademark Acknowledgements

Contents

Welcome to Core Objects

Firstly, thank you for either purchasing BlueFinity's Core Objects product or for taking the time to explore the great functionality that it can provide to you and your fellow developers.

This chapter outlines the members of the mv.NET family of products to which Core Objects belongs. It also summarizes the contents of this guide.

The mv.NET Family of Products

Core 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.

Core Objects not only provides the underlying framework upon which all other mv.NET products are based - it also provides, in its own right, a wealth of end-user capabilities to allow the developer to rapidly create feature-rich, high performance applications using the powerful tools provided by Microsoft's .NET environment.

The design goal of mv.NET is to enable the MultiValued developer to combine the power and flexibility of proven MultiValued technology with the state-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 Core Objects product provides a 100% native .NET interface to all major MultiValued database platforms, allowing .NET developers to access all aspects of MultiValued systems – both data and program code – from within their .NET application.

The Core 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 MultiValued developers to harness the full power of both their MultiValued system and the .NET platform.

Core Objects also has strong integration with Microsoft's Visual Studio.NET product, allowing the MultiValued developer to carry out virtually all aspects of application creation from within the VS.NET environment.

The product's key features are as follows:

- Feature-rich, MultiValued data structure aware data objects authored in 100% managed .NET code and supporting the native .NET databinding interfaces to promote rapid application development.

- High performance connections from client to database server using a variety of transport technologies dependent on flavor of MultiValued database platform.

- Advanced fetch-on-demand and background data retrieval technology, ensuring maximum application database performance

- Support for all major MultiValued platforms.

- Support for stateless applications, e.g. Web Services, featuring optimistic locking, automated state retention/reconnection and database connection pooling.

The mv.NET Suite

Core Objects is one of three products within the mv.NET suite; the suite comprising of:

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

- **Solution Objects** - Strongly-typed class-based access to your MultiValue database.

- **Adapter Objects** - complete implementation of an ADO.NET managed data provider for MultiValue databases, offering a standardized interface to database access.

Developer's Introductory Guide Contents

The contents of this guide are designed to allow a developer to easily install, configure and use the wealth of functionality provided by the Core Objects product. A summary of each chapter follows:

Product Installation

This chapter takes you through the process of installing Core Objects on both the developer's workstation and the MultiValued database server.

Technical Overview

This chapter lays out, in very broad terms, the full breadth of the technology provided by Core Objects. It is intended to be a 'gentle' introduction to the range of features that Core Object provides.

How do I ... ?

This chapter takes you through a series of 'How do I ...' style questions that act as a basis for learning the basics of using Core Objects. It is intended to be used as a framework for further, more detailed investigation using other chapters of this guide along with the product's comprehensive on-line help.

The Data Manager

The Data Manager is an important and valuable tool for the MultiValued developer. This chapter takes you through all of the features of the Data Manager and, in doing so, also explains how the Data Manager itself has utilized many of the aspects of the Core Objects class library.

Extended Dictionary Definitions

mv.NET supports the concept of extended dictionary definitions - meta data which allows a comprehensive definition of both the structure of data within a file and relationships between files to be created. This chapter explains both the content and relevance of this extended definition data.

The Session Manager

The Session Manager allows you to utilize Core Objects' session pooling capability. This chapter explains the concepts behind session pooling and takes you through the (very straight-forward) visual interface of the Session Manager application.

Class Library Overview

The object classes that make up Core Objects provide a wealth of functionality. This chapter outlines the hierarchy and content of the Core Objects class library.

Deploying Your Application

When the time comes to deploy your application, the relevant Core Objects components must be included within your setup process. This chapter discusses how this may be achieved.

The Sample Application

Core Objects is provided with a sample application which illustrates the use of many its features. This chapter takes you through this application, pointing out and explaining the key aspects of Core Objects that have been used.

All of the code examples in this guide use VB.NET syntax; however, the Core Objects components can be used with any .NET CLR compliant language, e.g. C# or J#.

Further Reading

In addition to the wealth of material contained within this guide, you may also find it very beneficial to read the 'Tips and Best Practices' guide which is also installed as part of the CID product. This guide contains many useful suggestions and insights into the ways in which mv.NET can be configured and used.

Support

If you have any queries about the installation or use of this product, please do not hesitate to contact BlueFinity's support staff who will be happy to assist.

email: support@bluefinity.com

Product Installation

This chapter outlines how to install the Core Objects product on both the developer's workstation and the MultiValued database host. **Please note that the Getting Started guide installed as part of the Client Interface Developer module contains a detailed set of instructions aimed at making the task of getting up and running with the product as smooth as possible**, the purpose of this chapter is to provide an overview of this process.

Installation Media

The product installation files can be downloaded from the BlueFinity ftp server. If you have already purchased the product, please email BlueFinity support (support@bluefinity.com) to request the current ftp site user name and password. If you would like to download a copy for evaluation purposes, please fill in the download request form on our website: http://www.bluefinity.com/downloads.asp

Product Installation Files

There are 3 product installations available:

CID – Client Interface Developer installation file
CRDK – Client Runtime Deployment Kit installation file
SRDK – Server Runtime Deployment Kit installation file

For most developers, simply running the CID setup program will be sufficient to install all the necessary development and runtime components onto your

workstation and will also install the necessary routines to allow you to download the server components onto your database server(s).

The CRDK and SRDK files are covered in the Deployment chapter within this guide. Note, the SRDK routine will automatically install a runtime version of the Data Manager as part of its installation procedure.

CID Installation File

The Client Interface Developer installation file will install the following components:

- Core Objects class library binaries
- mv.NET services
- Initial Configuration Database structure and content
- Server components download files
- Data Manager application
- Session Manager components
- Visual Studio.NET integration assemblies

The CID installation process requires very little intervention by the user – simply follow the on-screen prompts. At the end of the installation process, you will be able to start using the product.

After installing the CID. The basic procedure that you will need to follow to start using Core Objects on your development machine are:

1. Use the Data Manager utility to create a *Server Profile* within your *Configuration Database* and then download the server components onto your MultiValued system. Please refer to the following section and also the Server Console section within Data Manager chapter for details on how to do this.

   The Data Manager application (mvNET.DataManager.exe) is installed into:

   ```
   Program Files\BlueFinity\mv.NET\Version4.x\bin
   ```

   A shortcut to this program is created in the Start\Programs\mv.NET menu.

2. 'Enable' each of your application account(s) that you wish to access via mv.NET

3. Use the Data Manager utility to create an *Account Profile* within your *Configuration Database* for each 'enabled' account. At this point you should be able to connect into your account using the Data Manager to view and modify account data and program data.

4. Add a reference to the Core Objects assembly within your .NET application. This assembly (`BlueFinity.mvNET.CoreObjects.dll`) can be found in:

   `C:\Program Files\BlueFinity\mv.NET\Version4.x\bin`

Downloading Server Components

For Core Objects on the client to interact with a MultiValued database, the server hosting the database must have the mv.NET server components installed; this is done using the Data Manager.

Once you have installed the Client Interface Developer onto your workstation (which will automatically install the Data Manager application), you need to use the Data Manager to create a *Server Profile* representing the server onto which you wish to install the server components. Please refer to the section Creating a Server Profile within the Data Manager chapter for full details on how to do this.

Once you have created your server profile, a server node within the Data Manager's explorer tree will be created. Double-clicking on the *Server Console Window* node within this server node will allow you to open a session window onto the server, from which you will be able to initiate the download of the server components. Please refer to the Server Console section within the Data Manager chapter for full details on establishing a server console window.

Using the server console window's terminal emulator (or by using a terminal emulator of your own choice), you need to first create an account called MV.NET. On those MultiValued systems where a privilege level is associated with an account, the highest privilege level should be chosen. For MultiValued systems where an additional user logon is required, you should also create a user named MV.NET. For those MultiValued systems where a privilege level is associated with user logins, this user should be given the highest privilege level setting. You then need to logon to the MV.NET account (using the MV.NET user if relevant).

Strictly speaking, it is not mandatory for you to create a separate MV.NET account – you could install the mv.NET server components into any account, but for sake of clarity and understanding we recommend that you do create a dedicated account/user.

Once you are at command level within the MV.NET account you are ready to initiate the download of the server components. Select the *Action/Download Server Components* menu option from the session window's top menu bar. This will ask you to confirm that you are at command level with the MV.NET account and will then start the download process. The whole process should take about 5 or 6 minutes to complete.

Basically, all that the download does is to transmit the contents of 2 files onto the database server:

MVNET.INC
MVNET.BP

Both files contain the code of the server components. The download also creates a handful of additional ancillary files within the MV.NET account.

When the download has completed, a Notepad window holding the contents of the download trace file will be displayed. You need to glance down the contents of this file to make sure that all of the programs have been downloaded successfully. In the unlikely event of problems being encountered during the download, you will need to email the trace file to support@bluefinity.com. We will then advise you on what steps you need to take.

What Next?

After installing all the necessary Core Objects components, it may well be worthwhile spending a few minutes reading the following 2 chapters. These will provide an introduction to many of the more commonly used features of Core Objects.

Technical Overview

This chapter provides a technical backdrop to the Core Objects package. It discusses the main architectural aspects of the product and examines the main components that implement this architecture.

Core Objects

The architecture of Core Objects comprises three separate tiers:

- Client Interface Tier
- Session Pooling Tier
- MultiValued Server Tier

Each of these tiers is designed to be capable of residing on physically distinct systems separated by remote connections, but it may be that some or all tiers run on the same system – it all depends on the particular requirements of a specific installation.

The **Client Interface Tier** contains all the .NET programmer visible objects, i.e. the programming interface components that the developer works with in order to produce an application. It is very likely that most of your development effort will go into creating the contents of the client interface tier. The Client Interface Developer module of Core Objects contains many features to assist you in this task – see next section.

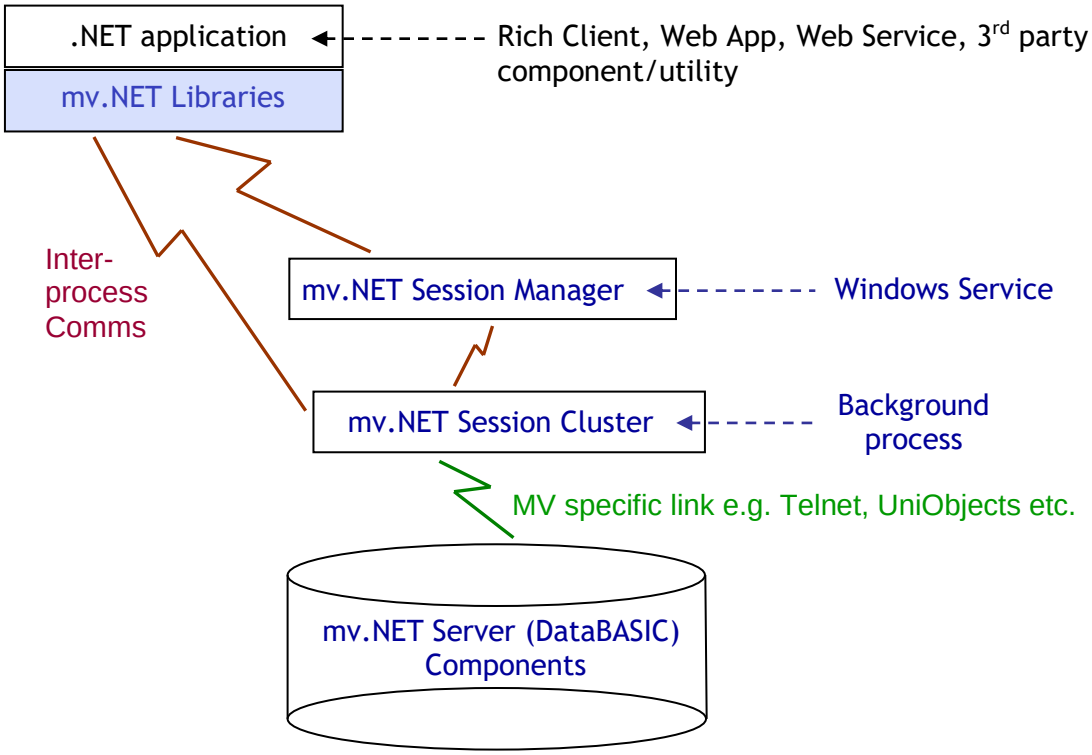
Typically, the Client Interface Tier will consist of the code that you write (using, for example, VS.NET), the Core Objects dll files that implement the Core Objects functionality plus any additional support files/assemblies that your application utilizes.

The **Session Pooling Tier** acts as the point of contact for Client Interface Tiers wishing to access one or more remote MultiValue systems. The Client Interface Tier to Session Pool Tier communications link utilizes .NET Remoting, which provides high a performance, loosely coupled connection to allow flexibility in the siting of these components.

The **MultiValue Server Tier** resides within the MultiValued database system. It is the point of contact for one or more Client Interface/Pooled sessions. The link to the MultiValued Server can utilize a variety of technologies, depending on what flavor of MultiValue system is being used.

The MultiValued Server tier is written in MultiValue DataBASIC and, therefore, a version of this tier (specifically coded and tuned) for each flavor of MultiValue database platform is provided.

The following diagram summarizes the interrelationships between these three tiers.



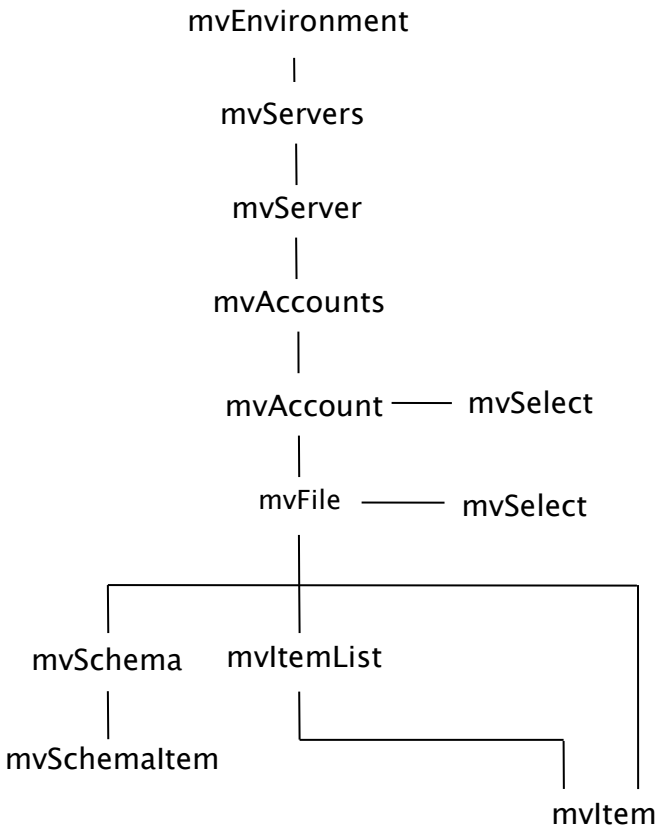
The Core Objects product consists of three separately installable packages:

- **Client Interface Developer**, which includes the Data Manager, Configuration Database and Session Pooling.
- **Runtime Deployment Kits**
- **MultiValue Server Module**

Client Interface Developer

The Client Interface Developer module should be installed on any workstation wishing to utilize Core Objects in conjunction with a development environment (such as Visual Studio.NET). It provides the programming interface into the Core Objects technology, along with plug-in extensions to Visual Sudio.NET to allow management of many aspects of a MultiValue system from within the VS.NET native IDE.

The primary feature of the Client Interface Developer is a class library holding the series of classes that allow a developer to connect to and work with a MultiValue database. The diagram below shows the basic hierarchy of the Core Objects classes:



As part of the Client Interface Developer module, an application (written using VB.NET and utilizing many aspects of Core Objects) is provided to allow the maintenance of MultiValue systems – such as file and dictionary creation. This application is known as the Data Manager. The Data Manager provides the following capabilities:

- Maintenance of the Configuration Database (see below)
- Account maintenance
- File maintenance
- Dictionary maintenance
- Item data maintenance

- Index maintenance
- Terminal emulation
- Intra and inter-system data transfer

The Data Manager is provided in the form of a standalone application.

To connect into a MultiValue system, Core Objects needs to know several things; namely:

- what MultiValue systems are available for connection
- the address of a particular system
- what communications transport medium to use when talking to a system
- how to login to a system

To that end, a Configuration Database is used to hold all of the following configuration information:

- The list (and details) of all known local MultiValue servers (Server Profiles)
- The list (and details) of all known accounts on all known local servers (Account Profiles)
- Session pooling configuration settings

The Configuration Database's contents are maintained using the Data Manager and setting up the initial contents of this database is one of the first tasks that you will need to do when starting to use Core Objects.

Each client application needs access to a Configuration Database (either sited locally or shared centrally over the LAN) to connect into a server.

As well as being able to maintain the contents of the Configuration Database using the Data Manager application, the Core Objects class library contains classes to allow the Configuration Database to be maintained programmatically by your own application code (this is, in fact, exactly what the Data Manager does).

Session Manager

The mv.NET Session Manager provides two key capabilities:

- To provide connection pooling across one or more individual processes or workstations. Note, the terms *connection pooling* and *session pooling* are used interchangeably within this guide.

- To monitor the presence and activity of an active database session

Connection Pooling is an essential feature for stateless applications, e.g. Web applications. It allows a database connection to be held open after an application has finished using it and then subsequently reallocated to another or same application instance at some future point in time - thus saving on the time and resources consumed in the initial creation of a database connection.

You should, therefore, utilize the Session Manager whenever there is a need to hold open database connections to avoid repeatedly incurring connection creation overheads.

The Session Manager can display a connection window for any active session, allowing the traffic flowing through that session to be monitored. For Telnet connections, a terminal emulator window is provided to allow keyboard generated input from the client to be entered.

As well as being essential for Web-based applications, session pooling is also very useful in a software development environment. In such an environment it is very typical for a programmer to continually start and stop their application as part of the debugging/testing cycle. On MultiValue systems where login can take several seconds to negotiate, the ability for the Session Manager to hold a disconnected session open, ready for re-use, is a valuable time-saving feature for the developer.

The Session Manager communicates with both application and session(s) via efficient inter-process mechanisms.

Runtime Deployment Kits

Runtime deployment kit (RDK) modules are required by an organization to enable deployment of the relevant mv.NET components necessary to run an application developed using Client Interface Developer licenses. The RDKs are supplied as part of Developer Licenses in two versions; one for client-systems where no local session pooling is required and one for server systems where session pooling is required. The relevant RDK will typically be incorporated within the developer's own product installation script and will install all the components needed to support the Core Objects class library at runtime.

A limited version of the Data Manager utility is also supplied as part of the runtime license.

Configuration Database

In order to connect into a MultiValued system Core Objects needs to know a number of things; namely:

- what MultiValued systems are available for connection
- the address of a particular system
- what communications transport medium to use when talking to a system
- how to login to a system

To that end, a *configuration database* is used to hold all of the following configuration information:

- The list (and details) of all known local MultiValued servers (Server Profiles)
- The list (and details) of all known accounts on all known local servers (Account Profiles)
- Session pooling configuration settings

The configuration database is held as a series and files and folders, the location of which is **CommonApplicationData**.

On modern systems, from Windows Vista onwards, this maps to physical folder:

```
C:\ProgramData
```

On older platforms, **CommonApplicationData** maps to physical folder:

```
C:\Documents and Settings\All Users\Application Data
```

Note, within Windows Explorer, the "ProgramData" folder may be hidden from view. However, you can browse to it explicitly using the top address bar input box within the Windows Explorer window.

The configuration database's contents are maintained using the Data Manager and setting up the initial contents of this database is one of the first tasks that you will need to do when starting to use Core Objects. Please refer to the How do I … ? and Data Manager chapters later in this document for more details on this topic.

Each client application needs access to a configuration database. This access is either via the Session Manager (in which case the client only needs to know the address and port number of the Session Manager) or via direct file access (either sited locally or shared centrally over the LAN).

As well as being able to maintain the contents of the configuration database using the Data Manager application, the Core Objects class library contains classes to allow the configuration database to be maintained programmatically by your own application code (this is, in fact, exactly what the Data Manager does).

MV Server Module

An essential part of Core Objects is the set of DataBASIC programs that reside on the MultiValued system. These programs are responsible for servicing all client generated requests and reside in an account typically named 'MV.NET' on the MultiValued database system.

These components are written in MultiValued DataBASIC and, therefore, a version for each flavor of MultiValued database is provided. Each of these versions has been specifically tuned for optimum performance on the target MultiValued platform.

The server components are downloaded onto the MultiValued system via the Data Manager application. Please refer to the Product Installation and Data Manager chapters for more details on this topic.

In order to access the data in an application account, the account has to be mv.NET enabled. This 'enabling' process creates several file pointers into the MV.NET account and also catalogs all of the server programs (if necessary). Again, please refer to the Product Installation and Data Manager chapters for more details on this topic.

How do I ... ?

This chapter covers a series of 'How do I ?' type questions. It is intended to be used as an introductory guide to getting started with the Core Objects product. If you have any 'How do I' style questions that do not appear in this chapter, please email them to us at support@bluefinity.com and we will be very happy to answer them. We will also add them to our on-line knowledge base for the benefit of others.

Many of the answers provided in this chapter refer to classes within the Core Objects class library. Please refer to the Class Library Overview chapter along with the on-line help for much more in depth information on these matters.

How do I start using Core Objects?

The first action is to install the software - please refer to the previous chapter for full details on how to do this. The software installation process needs to be done on your software development workstation - a workstation that must, at the very least, have the Microsoft .NET framework installed, but will typically also have Visual Studio .NET installed.

The Core Objects module which needs to be installed first is the *Client Interface Developer*. The installation routine for this product will install all of the components that you will utilize in getting connected to and then working with a MultiValued database.

The CID setup routine installs a Getting Started Guide. We strongly recommend that developers read this guide if they are new to mv.NET. A link to the guide is placed in the Start\Programs\mv.NET menu.

Once the Client Interface Developer has been installed, you need to reference the following dll from within your .NET project:

`BlueFinity.mvNet.CoreObjects.dll`

This dll can be found in **Program Files\BlueFinity\mv.NET\bin\version**

Note, in Visual Studio .NET, you can use the Project\Add Reference menu option to add project references.

Once you have installed the software, you then need to use the Data Manager to define a server and an account profile to allow access to your MultiValued account data and programs. See next How do I ?

How do I let Core Objects know about the databases that I wish to access?

Before Core Objects can connect to a MultiValued database, you need to give it some basic details about where the database is located, how it can be connected to and what kind of flavor it is. These pieces of definition information are collectively known as a 'Server Profile'. Thus, you need to create a server profile for each MultiValued database installation that you wish to access.

The easiest way to do this is to use the *Data Manager* application – this will have been installed as part of the Client Interface Developer product installation procedure. A shortcut to the Data Manager will have been placed in your Start\Programs\mv.NET menu.

Please refer to the Data Manager chapter for details on how create a server profile.

Once you have created a server profile, you will need to create an 'account profile' for each MultiValued database account that you wish to access on that server. Again, the Data Manager should be used to do this.

Please refer to the Data Manager chapter for details on how create an account profile.

Once you have a server profile created that contains an account profile, you are ready to connect to and use your MultiValued database.

How do I connect to a database from my application?

The first thing to check is that you have a reference to the following Core Objects assembly (dll file):

```
BlueFinity.mvNet.CoreObjects.dll
```

This dll can be found in **Program Files\BlueFinity\mv.NET\bin\version**

Next, you need to create an instance of an mvAccount object. This can be done as follows:

```
Dim myAccount As New mvAccount(LoginProfileName)
```

The Login variable needs to be the name of a Login Profile that has been defined using the Data Manager. We recommend that you use Login Profiles wherever possible as it provides a valuable level of naming abstraction from the names of your servers and accounts. Please refer to the Creating a Login Profile section within the Data Manager chapter for further details on Login Profiles.

An important aspect of the login process is the mechanism by which mv.NET locates the configuration database (the place where your server, account and login profiles are stored). The process (in order of sequence) which mv.NET goes through in order to ascertain the location of the configuration database is as follows:

1. If address of the Session Manager has been passed in via either the mvEnvironment.Login method or the mvAccount constructor, use that information.
2. If path of the configuration database has been passed in via either the mvEnvironment.Login method or the mvAccount constructor, use that information.
3. Look for the 'ConfigurationPath' file in CommonApplicationData. If it exists, read its contents and look for a 'Configuration' folder at that location.
4. Look for a 'Configuration' folder in CommonApplicationData. If it exists, use that as the configuration database

If none of the above succeeds an error is raised.

An example of the address of the Session Manager being passed as part of the mvAccount constructor is as follows:

```
Dim serverAddress As String = "Pluto:10013"
Dim myAccount As New mvAccount(LoginProfileName, ServerAddress)
```

Here, Pluto is the system name of the server hosting the Session Manager – listening on port 10013 (which is the default port).

How do I open a data file and read items?

Using an mvAccount object (see 'How do I connect into a database?'), you first need to use its *FileOpen* method, for example, to open a file called ORGANIZATION, you would use the following code:

```
Dim OrgFile As mvFile = myAccount.FileOpen("ORGANIZATION")
```

If the file is successfully opened, the mvFile object's Read Method can then be used to read an item. For example, to read item ID '0001', you would use the following code:

```
Dim OrgItem As mvItem = OrgFile.Read("0001")
```

The mvItem object can then be used to access the data within the item. See 'How do I access the data within an item?'.

Note, the mvFile object supports a range of methods that read data from a file, e.g. ReadV and ReadBool. Please refer to the class library chapter for further details.

How do I access the data within an item?

Using an mvItem object (see 'How do I open a data file and read items?'), you need to use its *Data* property in order to both retrieve and update its data content. For example, to set the Text property of a control to attribute 3 of an item, you would use the following code:

```
txtAddress.Text = OrgItem.Data(3)
```

The Data property is the default indexer of the mvItem class, therefore you could shorten the above code to:

```
txtAddress.Text = OrgItem(3)
```

In C# (because the language has a less flexibility in its support for properties) you would have to use the following code:

```
txtAddress.Text = OrgItem.getData(3);
```

As well as accepting an integer argument, the Data property can alternatively accept the name of a dictionary item for the file, So, using the previous example, if we assume that dictionary item 'ADDRESS' is defined to access attribute 3, the following code could also be used:

```
txtAddress.Text = OrgItem("ADDRESS")
```

The Data property can additionally be passed a multivalue and subvalue position to further refine the piece of item data that is accessed. For example, if we assume that the address field is MultiValued, the following code would retrieve the 3rd multivalue from the ADDRESS attribute.

```
txtAddress3.Text = OrgItem("ADDRESS", 3)
```

Note, that when data is retrieved using a dictionary name, the mvItem object will return a value that has been converted (if relevant) and cast as the appropriate data type using the contents of the dictionary item to guide it in this conversion/casting process. The exception to this rule is if a return value contains multivalue or subvalue marks, in which case constituent values will be converted to their appropriate output format but the overall return value must be cast as a string type.

Retrieving data via attribute position, on the other hand, will always result in an unconverted (raw item) string value being returned (unless you explicitly supply a conversion code as an argument to the Data property call).

Updating the data within an item is just as straightforward. For example, to amend attribute 1 in an item, the following code would be used:

```
OrgItem(1) = "XYZ Systems Inc."
```

Or, using a dictionary name:

```
OrgItem("NAME") = "XYZ Systems Inc."
```

In C#, you need to use the setData function:

```
OrgItem.setData(1,"XYZ Systems Inc.");
```

The same rules (in terms of data conversion and casting) as per retrieving data content via dictionary name apply to updating data via dictionary name. That is, the following code would result in attribute 4 being set to string "Feb 13 1961":

```
OrgItem(4) = "Feb 13 1961"
```

Whereas, the following code would result in attribute "DATEFORMED" being set to the multivalued database internal date integer value representing date Feb/13/1961 (if the dictionary item DATEFORMED contains a date conversion/correlative).

```
OrgItem("DATEFORMED") = "Feb 13 1961"
```

If you want to update data content via attribute position, but still want data to be stored in internal format (i.e. input conversion to be performed), you will need to pass a conversion code as part of the update call. The following examples illustrate storing a date value and a currency value:

```
OrgItem(4, "D") = "Feb 13 1961"
OrderItem(3, "MR2") = 2050.75
```

Finally, the mvItem list object also has an *ID* property which returns the associated item ID.

How do I write item data back to a file?

Using an mvItem object (see 'How do I open a data file and read items?'), you need to use its *Write* method in order for its data content to be written back to the database. For example:

```
OrgItem.Write
```

The above line of code will write the current data content of the OrgItem to the associated file and item ID that were used to retrieve it originally.

If you wish to write an item back to a different ID in the same file, you can supply an alternative ID as an optional argument to the Write method:

```
OrgItem.Write("0002")
```

If you wish to write the item back to different file, you need to use the Write method of the alternative mvFile object. For example:

```
OrgArchiveFile.Write OrgItem
```

How do I delete items from a file?

You can delete items from a database file by using either an mvFile object or an mvItem object.

Using an mvFile object, you need to use its *Delete* method. For example, to delete item "0001" from the file which has been opened into variable OrgFile you would use the following code:

```
OrgFile.Delete("0001")
```

With an mvItem object, you must use its *DeleteItem* method. This will delete the item indicated by the value of the object's *ID* property. Note, the object's data content (i.e. the data content of the object in client memory) will be unaffected by the use of the DeleteItem method – only the database file item will be deleted.

How do I run my DataBASIC programs?

There are 2 ways of executing DataBASIC code from within your .NET application – the method you should use depends upon whether the code is within a main-line program or a subroutine.

If it is a main-line program, you can execute it by using the *Execute* method of an mvAccount object. For example, to run program 'BATCHUPDATE' you would use the following code.

```
myAccount.Execute("BATCHUPDATE")
```

In fact, the Execute method can be used to run any command-level statement. You can also provide arguments to the Execute call that will capture any output produced by the command and also its return status.

If your code resides in a subroutine, you need to use the *CallProg* method of the mvAccount object. This method also allows you to pass multiple (updateable) arguments to the subroutine. For example, to call subroutine CHECKORDER, passing 2 arguments you would use the following code:

```
myAccount.CallProg("CHECKORDER", OrderNo, OrderStatus)
```

It is very important to make sure that the number of arguments that you supply to the CallProg method exactly matches the number of arguments defined within the subroutine that you are calling.

How do I select items from a file?

There are a number of ways of selecting data from a file. Below are the most common:

mvAccount.Select
mvFile.Select
mvFile.IndexSelect
mvFile.QSelect
mvAccount.ProgSelect

The first 2 methods allow you to select items from a file in a manner which allows you to specify selection and sort criteria. The third method (IndexSelect) allows you to select items from a file using an index which has been associated with the file. The fourth method (QSelect) allows you to select items based on item IDs held within a specific item within a specific file. The final method (ProgSelect) allows you to invoke your own DataBASIC subroutine in order to provide the list of qualifying items IDs for the selection.

Below are examples of each of the above methods being used:

```
myAccount.Select("SELECT ORGANIZATION BY NAME WITH NAME = ""E]""")

OrgFile.Select("NAME = ""E]""", "BY NAME")

OrgFile.IndexSelect("NAME", "SW", "E")

OrgFile.QSelect("PRODUCT", "RGY954", "4")

myAccount.ProgSelect("MYSUBROUTINE", "HEAVY", PRODUCT)
```

The first 2 examples result in exactly the same selection of items, i.e. all organizations with a NAME attribute starting with the letter 'E' sorted in ascending NAME order.

The third example assumes that you have created an index for the file called 'NAME' which indexes the ORGANIZATION file on sorted NAME attribute. It selects items from this index starting at the point where NAME starts with the letter 'E' and will stop selecting when a name not starting with 'E' is encountered.

The fourth example assumes that a MultiValued list of supplier codes is held in attribute 4 of all product items. The code example here, thus, reads the list of item IDs from attribute 4 of item RGY954 within the PRODUCT file and then uses this as the basis to assemble a list of items from the ORGANIZATION file.

The fifth example results in the subroutine "MYSUBROUTINE" being called in order to return which items from the PRODUCT file are to be selected. The data string of "HEAVY" (this could be any string data) is passed into the subroutine in order to indicate the context of the call.

All of the methods that can be used to select items from a file return an *mvItemList* object. This object can then be used to access the selected items. The following example illustrates how to access the first attribute of the 3rd selected item with mvItemList variable orgItems:

```
name = orgItems(3)(1)
```

If you want to iterate through the selected items, you can use the mvItemList's EOL (end of list) property:

```
Do Until orgItems.EOL
    OrgItem = orgItems.ReadNext
Loop
```

Please refer to the class library chapter for more details on the mvItemList object. Note, that the ReadNext method (as used above) returns an mvItem object – not just an item ID. You can then use the *ID* property of this item if you wish to access its item ID. However, if you ONLY want the item ID, the ReadNextID method is more efficient.

Finally, all of the Item selecting methods can accept an *mvSelect* object within their argument list. The mvSelect object provides a dozen or so properties that provide a high degree of control over both what data is selected from the server and how the selected data it is passed back to the client. Some examples of these properties are:

DictionaryList – the list of dictionary derived values required
AttributeList – the subset of attributes required
PreSelection – the command to run before the main selection
RetrievalStyle – the style of data retrieval from the server to client

One of the main reasons for providing the mvSelect object is to allow the developer to optimize both the volume and timing of data transfer from server to client – this issue being of prime importance when ensuring that applications are scalable and capable of being run across a variety of network bandwidths.

Please refer to the class library chapter for more details on the mvSelect object.

The Data Manager

The Data Manager is an important and useful tool for the application developer. In order to use Core Objects you will need to learn a little about how to use it.

This Chapter explains each aspect of the Data Manager. It starts by covering those parts which you will definitely have to use in order to start utilizing Core Objects (Configuration Database maintenance) and then follows on from this to cover features designed to improve programmer productivity when designing and developing server-resident data structures and programs.

Note, the Data Manager is also a classic example of an application making use of the Core Objects class library. All its database interaction functionality is achieved using the Core Objects class library. The Data Manager is written using VB.NET.

Data Manager Versions

The Data Manager is provided in two different forms:

- Standalone full version
- Standalone runtime version

The full versions support all the available Data Manager functionality, the runtime version supports a subset of this functionality and is designed to be distributed as part of your application deployment rollout when required. Note, the SRDK setup routine will automatically install a runtime version of the Data Manager as part of its installation procedure.

Running the Standalone Data Manager

The CID setup installation file will place the Data Manager executable (mvNET.DataManager.exe) in the following directory:

```
Program Files\BlueFinity\mv.NET\Version4.x\bin
```

It will also create a shortcut to this program in the Start\Programs\mv.NET menu.

The standalone Data Manager is an MDI application. When you start it up, the left hand side of the main MDI form displays a treeview list containing a number of entries; this is known as the Data Manager *Explorer*. An example is shown in the following screenshot

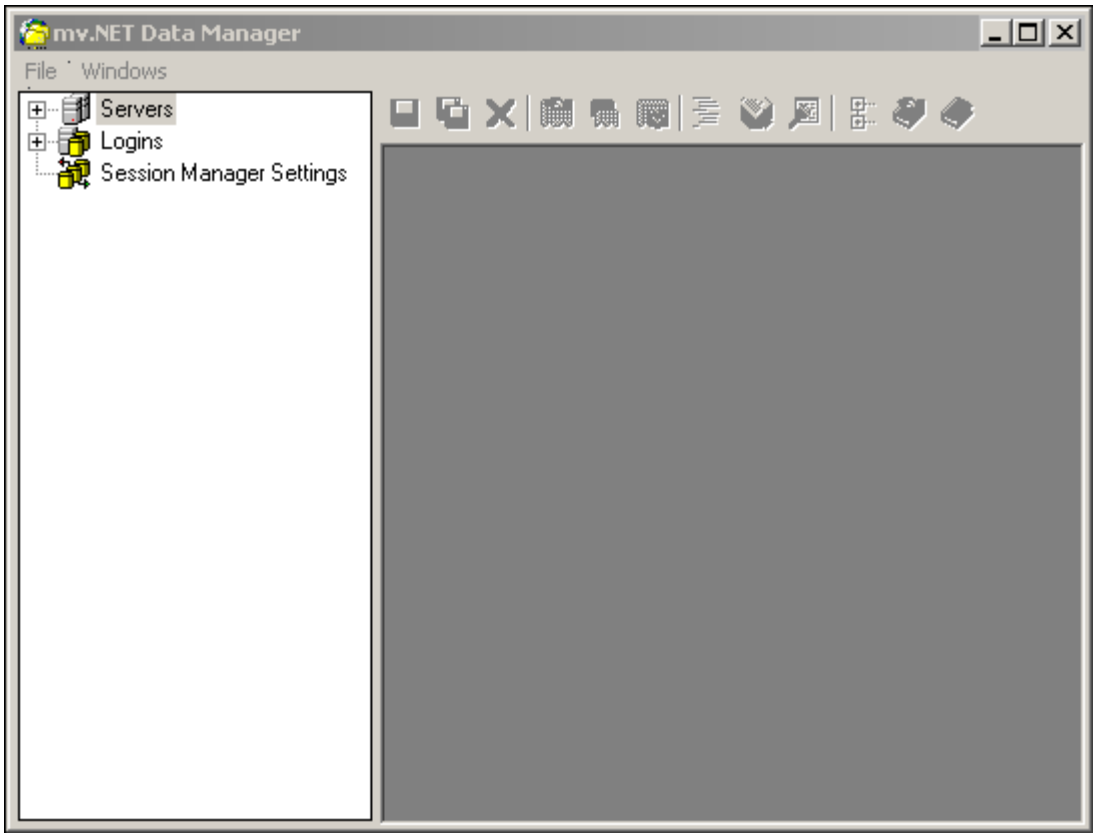


Diagram 2a : The Data Manager Explorer

The Explorer tree is used like any normal treeview control, with most of its nodes supporting context sensitive right-click popup menus.

Each of the node types is explained in the following sections.

Explorer Nodes : Overview

The higher level nodes of the Explorer tree allow you to maintain the contents of the Configuration Database. As you move down through the hierarchy of the Explorer nodes, you are able to establish database sessions and maintain the contents of MultiValued databases. The following section explains the Configuration Database in detail

Most of the nodes within the Explorer treeview support right-click popup context menus. These allow you to perform a wide variety of tasks. Each of the explorer node sections below explain the options available within these popup menus.

The Configuration Database

The Configuration Database (CDB) is a vital aspect of Core Objects. It holds the information which allows Core Objects to be aware of which MultiValued systems are available for connection and how to connect into these systems.

The CDB is a collection of directories and files which, by default, will be located in in CommonApplicationData. On Vista, Windows7 and Server2008 systems, **CommonApplicationData** maps to physical folder:

```
C:\ProgramData
```

On all other platforms, **CommonApplicationData** maps to physical folder:

```
C:\Documents and Settings\All Users\Application Data
```

Note, within Windows Explorer, ProgramData may be hidden from view. However, you can browse to it explicitly using the top address bar input box within the Windows Explorer window.

However, if you wish to share a configuration database amongst a population of users or developers, you can create a file in the above location called ConfigurationPath which points to the actual location of the CDB. The ConfigurationPath file should be a simple text file holding one line of text; this one line should contain the fully qualified path of the location of the CDB. Note, the path within the ConfigurationPath file should NOT include the '\Configuration' at the end; i.e. it should be the path of the folder containing the 'Configuration' folder.

Also note that the constructor of the mvEnvironment class also allows you to specify the location of the CDB.

An alternative approach to specifying the location of the CDB is to supply the address and port number of the Session Manager. The Session Manager is installed as part of the CID setup and SRDK setup routines. The syntax of the string which needs to be supplied via either the contents of the ConfigurationPath or via the mvEnvironment constructor is:

```
Address:Port    e.g.  Pluto:10013
```

Where *Address* is either the IP address or resolvable system name of the server which is hosting the Session Manager and *Port* is the port number on which the Session Manager is listening. These settings can be maintained via the Data Manager, see the Configuring Session Management section in the The Session Manager chapter.

Within the CDB, there are 4 main categories of information:

Server Profiles
Account Profiles
Login Profiles
Session Management Profile

Server Profiles contain most of the information which allows Core Objects to contact and connect into a MultiValued database system. There should be one server profile for every database installation that you wish to access.

Within a Server Profile you may define many Account Profiles; one for each account that you wish to access via mv.NET within the specified server. An Account Profile holds several pieces of data that can be used in establishing a connection to the account along with various pieces of control data allowing the behavior of session pooling and temporary data housekeeping to be controlled.

Login Profiles provide a means of associating a logical name with a server/account pairing. This allows you to avoid 'hard coding' server/account names into your application, which is important especially to software houses developing applications for resale.

The Session Manager Profile allows you to control the behavior of the Session Manager. See the Session Manager chapter for full details on this topic.

The following sections take you through the creation and maintenance of all these profiles.

Explorer Node : Servers

The Servers node of the explorer contains one node per server profile. The right-click popup context menu of this node contains the following options:

Open

Expands the Servers node to allow the list of server profiles to be viewed

Add Server Profile

Allows a new server profile to be created. See section Creating a Server Profile.

Paste Server Profile

Allows a new server profile to be created based on a previously copied profile. See menu option Copy Server Profile within the Server node context menu.

Explorer Node : {server profile name}

Each server profile node of the explorer contains an *Accounts* node (which contains the list of account profiles defined within this server profile) and a node for opening up a server console window onto the server. The right-click popup context menu of this node contains the following options:

Open

Expands the server profile node to allow the list of sub-nodes to be viewed

Edit Server Profile

Open the server profile maintenance window.

Rename Server Profile

Allows the name of the server profile to be changed.

Copy Server Profile

Places the server profile definition onto the Explorer's clipboard ready for pasting.

Cut Server Profile

Places the server profile definition (in cut mode) onto the Explorer's clipboard ready for pasting.

Delete Server Profile

Removes the server profile from the configuration database, removing all constituent account profile definitions.

Explorer Node : Accounts

The Accounts node contains a node for each account profiles defined within the parent server profile. The right-click popup context menu of this node contains the following options:

Open

Expands the Accounts node to allow the list of account profiles to be viewed

Add Account Profile

Allows a new account profile to be created. See section Creating an Account Profile.

Paste Account Profile

Allows a new account profile to be created based on a previously copied profile. See menu option Copy Account Profile within the account node context menu.

Explorer Node : Logins

The Logins node of the explorer contains one node per login profile. The right-click popup context menu of this node contains the following options:

Open

Expands the logins node to allow the list of login profiles to be viewed

Add Login Profile

Allows a new login profile to be created. See section Creating a Login Profile.

Explorer Node : {login profile name}

Each login profile node of the explorer has a right-click popup context menu containing the following options:

Edit Login Profile

Open the login profile maintenance window.

Rename Login Profile

Allows the name of the login profile to be changed.

Copy Login Profile

Places the login profile definition onto the Explorer's clipboard ready for pasting.

Cut Login Profile

Places the login profile definition (in cut mode) onto the Explorer's clipboard ready for pasting.

Delete Login Profile

Removes the login profile from the Configuration Database.

Explorer Node : Server Console Window

The server console window node allows you to open a server console window on the parent server. The right-click popup context menu of this node contains the following options:

Open Server Console Window

Displays a window asking you to confirm the server connection address and then opens the console window. Please refer to the section Server Console Window for details on what you can do within a server console window.

Explorer Node : {account profile name}

Each account profile node of the explorer contains a series of sub-nodes representing the contents of the associated database account. These sub-nodes can only be displayed once a login session onto the account has been established -

see menu option Login below. The right-click popup context menu of this node contains the following options:

Login
Initiates a login onto the account. The section Creating a Server Profile contains a description of the login process.

Logout
Terminates the currently active database session.

Edit Account Profile
Open the account profile maintenance window.

Rename Account Profile
Allows the name of the account profile to be changed.

Copy Account Profile
Places the account profile definition onto the Explorer's clipboard ready for pasting.

Cut Account Profile
Places the account profile definition (in cut mode) onto the Explorer's clipboard ready for pasting.

Delete Account Profile
Removes the account profile from the configuration database.

Explorer Node : Session Manager Settings

This node allows you to maintain the settings of the Session Manager on the system. Note, this is only relevant if you have installed the Session manager module. The right-click popup context menu of this node contains the following option:

Maintain Settings
Displays a window allowing you to activate/deactivate session management on this system and also allows you to specify the address of the system which is hosting

the session manager service . Please refer to the Session Manager chapter for further details on maintaining these settings.

Creating a Server Profile

To create a new server profile, select the Add Server Profile option from the Servers context menu. This option will prompt you for a profile name. After entering a name, the following window will be displayed allowing you to enter the details of your new profile:

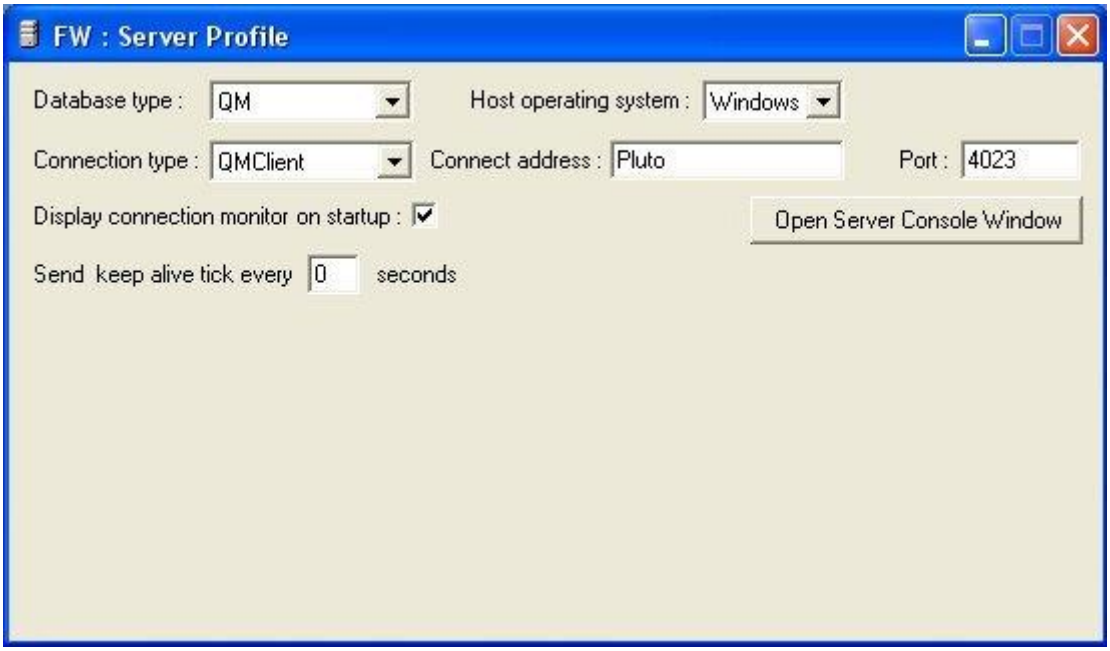


Diagram 5b : The Server Profile Maintenance Window

The various fields on this form are explained below:

Database type : This allows you to specify the 'flavor' of MultiValued platform that this database installation represents.

Host operating system : This allows you to specify the operating system running on the server.

Connection type : This allows you to specify the type of connection that needs to be established with the server. The options available here will vary depending on the *Database type* selected.

Connection address : This should be set to the address of the server. This can be either an IP address or a resolvable system name.

Port : This allows you to (optionally) specify the port of the listening service for the specified connection type. See next section for more details. For some connection types this field is not required and will be hidden.

Display connection monitor on startup : If this field is ticked, a connection monitor window will be automatically displayed when a new connection to this server is established. **Note, the Session Monitor window must be open for this to occur**.

Send keep alive tick : This input field allows you to indicate that traffic is to be automatically generated between client and server in order to avoid session inactivity timeout from occurring. By specifying a non-zero number in this field, a small number of characters will be transmitted from client to host every so often in order to prevent the communications link from timing out.

The **Open Server Console Window** button in the top section of this window will open a terminal emulation window onto the server. Please refer to the next chapter for more details.

Connection Port Specific Details

Below is a table detailing the way in which the Port field should be used depending on the selected connection type.

Connection Type	Field	Description
IP	Port	The port number of the Telnet listener. Defaults to 23 if left blank Or The port number of the SSH listener, prefixed by the letters "ssh", e.g. "ssh22" indicates an SSH-based connection on port 22.
UniObjects	Service	The name of the UniObjects listening service. Defaults to 'udcs' for Unidata and 'uvcs' for UniVerse if left blank. Note, the 'Port' prompt changes to a 'Service' prompt for UniObjects-based definitions.
SAC	Port	The name of the UniVision service. Defaults to

		'UniV' if left blank.
D3ODBC	Port	This field can contain 3 discrete pieces of information, each separated by a colon : ***VM_name*:Port:*D3_version*** Where *VM_name* is the name of the D3 virtual machine; *Port* is the port number on which the D3 ODBC service is listening and *D3_version* is the version classification of the D3 server. If this field is left blank, the follow default value is used: ***Address*:1603:754** For Windows hosted D3 servers, *Address* is set to the same value as the Connection address field; for UNIX-based systems, it is set to "pick0". If only one value is entered into this field, if it is non-numeric it is assumed to be the virtual machine name - otherwise it is assumed to be the port number.

Defining Connection Negotiations

Within the server profile definition form, if you select a connection type (for example 'IP') that requires a negotiation sequence to be followed in order to gain access to the database account, an extra region of input fields will be presented in the first tab of the form:

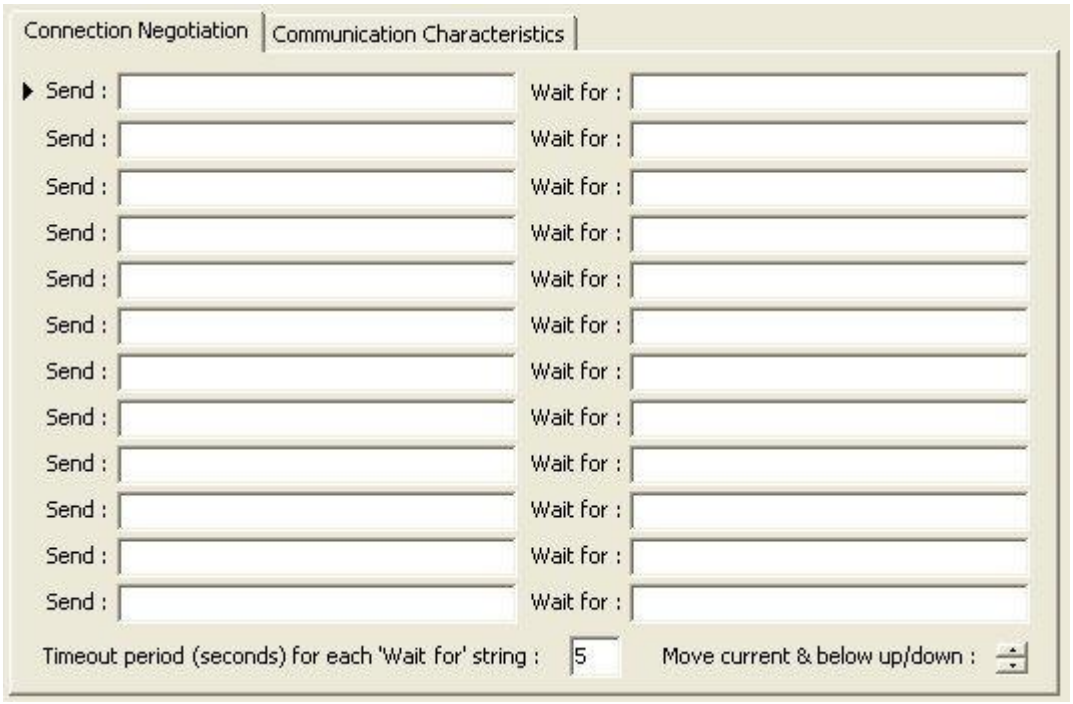


Diagram 5c : Connection Negotiation Fields

The various fields on this form are explained below:

Send/Wait for: These input fields allow you to specify the character strings to be sent to the server in order to gain access to the required account. After each send string you may specify a character sequence that must received within the stream of characters from the server before sending the next send string. In such a way you are able to correctly synchronize the transmission of send strings to the host.

Within the Send and Wait for input fields you may include various special 'marker' strings:

{account} – will insert the *Account* field from the relevant account profile. When you logon to a server you will always need to specify both a server and an account profile name. The account profile definition contains an *account* field – it is the value of this field which is inserted in place of the {account} marker at run time.

{user} – will insert the *User* field from the relevant account profile.

{password} – will insert the *Password* field from the relevant account profile.

{password#2} – will insert the *Password#2* field from the relevant account profile.

{prompt#1} – will insert the *Prompt#1* field from the relevant account profile.

{prompt#2} – will insert the *Prompt#2* field from the relevant account profile.

{sleep} – will cause the connection negotiation process to sleep for 1 second. The word sleep may be followed by a space and then an integer number to sleep for more than one second; e.g. {sleep 3} will cause a sleep for 3 seconds.

~n – will insert character n; e.g. ~13 will insert a carriage return character.

Note that the Wait for strings ***are*** case sensitive.

The send/wait for pairings need to navigate the logon process down to command level within the target account, at which point the (final) send string of 'MVNET.START~13' should be sent with no wait for after it. This command starts the mv.NET IP listener process, after which, mv.NET will take over the negotiation process and eventually place the listener into a 'ready' state.

Timeout period: Allows you to specify the maximum number of seconds that a Wait for string will be waited for. If a Wait for string does not appear in the received characters buffer within this number of seconds after its associated send string has been transmitted, the login process will be abandoned.

It is, obviously, important to get the send and wait for strings absolutely correct, otherwise Core Objects will get lost part way through the connection process and will fail to establish a link to the server. To that end, there is a button on the server profile maintenance window named 'Open Server Console Window'. Clicking this button will launch a simple terminal emulation window which, amongst other things, will allow you to observe the exact sequence and character content of the series of prompts and messages that the MultiValued system displays during the connection process.

The spin up/down button in the bottom right of this window allows you to insert and remove lines mid-way within the send/wait pairs. The arrow-head in the left-hand edge of the tab indicates the "current" row.

Connection Control

At the foot of the connection negotiation tab is an area which allows you to specify settings that control the creation of new connections to this server:

Diagram 5d : Connection Control Settings

The first 3 input fields allow you control the suspension of new connections to this server based on the rate of new connection attempt failures.

The last 2 input fields allow you to restrict the rate of new connection attempts based on a fixed allowance for a specified period of time.

The purpose of these input fields is to allow you to control the behaviour of the Session Manager in situations where a database server is encountering network connection problems or where it is under a high degree of stress. In such situations it is common for connections to the database server to become unstable and so the above settings allow you to restrict the impact that mv.NET has on the database server. That is, in such situations, usually the last thing that the database server needs is extra workload associated with trying to establish new connections continually.

Communication Characteristics

In addition to specifying connection negotiation details, if a connection type is negotiated a second tab allowing the definition of communication characteristics:

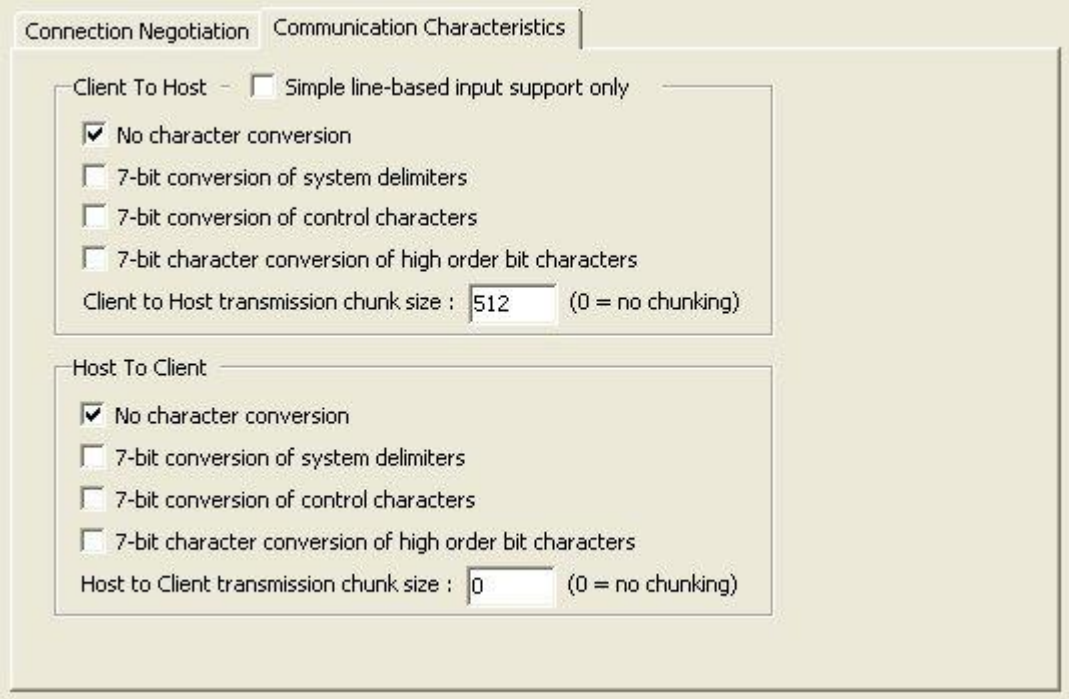


Diagram 5e : Communication Characteristics

The various fields on this form are explained below:

Client to Host: These input fields allow you to specify the nature of the communications link to the server in terms of data flow from client to server (host). The check boxes allow you to define 7/8 bit characteristics, the **chunk size** input field allows you to specify the maximum number of bytes that may be transmitted in a single uninterrupted burst to the host.

Host to Client: These input fields allow you to specify the nature of the communications link to the server in terms of data flow from the server (host) to the client system. The check boxes allow you to define 7/8 bit characteristics, the **chunk size** input field allows you to specify the maximum number of bytes that may be transmitted in a single uninterrupted burst from the host.

The following table describes the effect of ticking each checkbox

Checkbox Name	Effect
No character conversion	All characters are transmitted without any form of conversion being applied.
7-bit conversion of system delimiters	All characters in the ASCII range 250-255 are converted into 7-bit forms.

7-bit conversion of control characters	All characters in the ASCII range 1-31 are converted into 7-bit forms.
7-bit conversion of high-order bit characters	All characters in the ASCII range 128-255 are converted into 7-bit forms.

Note, the following scenarios may require the use of character conversion:

- situations where IP connection types are being affected by stty and other terminal control settings on the database host
- where the Windows system hosting the mv.NET Session Manager service is configured to use an Eastern European or Slavic (Cyrillic alphabet) based language setting
- NLS support is being used on a U2 database

Finally, the "Simple line-based input support only" checkbox allows you to indicate whether the database platform only supports old-technology line-based input capabilities. This should usually be left unticked unless you have specific reasons to believe that it applies to your particular database server.

Creating an Account Profile

Each server profile holds the definition of the main aspects controlling how Core Objects clients locate and connect into a MultiValued database server. However, this definition is not complete, as it needs augmenting with the details of a particular account on the server so that an account specific connection can be established. The Account Profile is the entity which holds this extra information and there may be several account profile definitions within a single server profile - each representing a different account or application on the server.

An account profile provides the means to define 2 main areas of information. Firstly, account specific logon settings and secondly, account specific session pooling and housekeeping settings.

To create a new account profile, select the Add Account Profile option from the Accounts context menu. This option will prompt you for a profile name. After entering a name, the following window will be displayed allowing you to enter the details of your new profile:

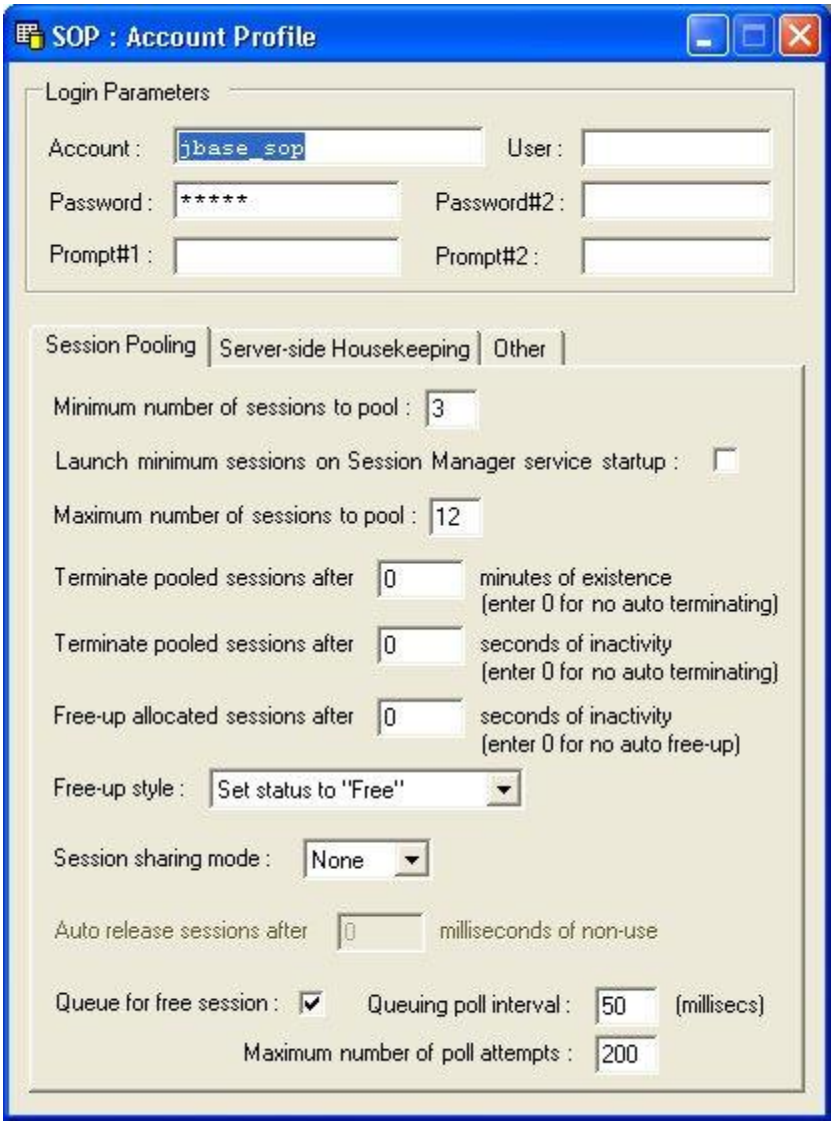


Diagram 5f : The Account Profile Maintenance Window

The various fields on this form are explained below:

Login Parameters

This group of input fields allows you to specify data that can be used to control Core Objects' login process to correctly connect to the required account. The values of these 6 fields can be inserted into connection negotiation send and wait for strings at run-time using special markers – see previous section Defining Connection Negotiations.

These fields, combined with the server profile definition, thus provide a complete definition to mv.NET of how to connect into a specific account on a specific server.

The table below lists how these 6 fields can be utilized for the variety of connection types which may be used within a server profile:

Connection Type	Description
IP	The 6 Login Parameters fields may be inserted into the send and wait for strings of a server profile's connection negotiation definition using the special markers {account}, {user}, {password}, {password#2}, {prompt#1 and {prompt#2}.
UniObjects	(UniVerse and UniData only) The *Account* field should be set to the path of the directory holding the required account. The *User* field should be set to the Windows/Unix user that is to be used for the login process. *Password* should be set to the user's password.
QMClient	(QM only) The *Account* field should be set to the required QM account. The *User* field should be set to the Windows/Unix user that is to be used for the login process. *Password* should be set to the user's password.
SAC	(UniVision only) The *Account* field should be set to the name of the database account that is to be connected into.
D3ODBC	(D3 only) The *Account* field should be set to the name of the database account that is to be connected into. It is assumed that name of the account holding the mv.NET server-side routines is "MV.NET". If this is not the case, the actual name of the account can be supplied at the end of the account name, separated by a colon, e.g. **SOP:mvNET** Where "mvNET" is the name of the account into which the server-side code of mv.NET was downloaded.

Session Pooling

This group of input fields allows you to specify session pooling settings for this particular account. Please refer to the Configuring Session Management section within the Session Manager chapter for details on how these fields may be used.

Housekeeping Settings

The second tab on the account profile maintenance window allows you to define the server-side housekeeping activities that mv.NET can perform, as shown below:

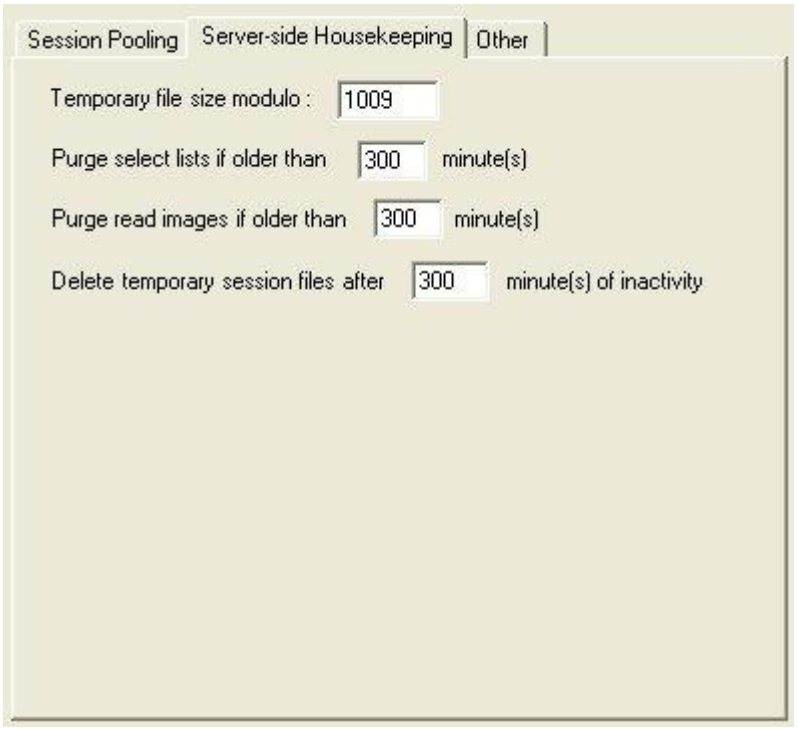


Diagram 5g : Server-side Housekeeping Settings

All housekeeping activity is coordinated by the session manager service and thus the frequency of housekeeping is set at the session manager level. This can be done by double-clicking the Session Manager Settings node within the Data Manager treeview. At the foot of the resulting settings window you can specify both the timing method for housekeeping (either to be run every x minutes or at a specific time each day) and the interval/time setting itself.

The account profile housekeeping settings allow you to control when temporary data on the server is to be deleted, as well as controlling the default size of temporary files created automatically by mv.NET.

The *Purge select lists* setting allows you to define when server-side select lists are to be deleted. When a select list is created by mv.NET as part of its internal activity (which will, of course, have been triggered by some application request) it is given a timestamp. When the housekeeping process executes, it examines the timestamps of all select lists created by mv.NET for this account and will delete any that are older than the age specified within the account profile.

The *Purge read images* setting allows you to define when optimistic lock read images taken on the server are to be deleted. When an item is locked in optimistic mode a copy (read image) of the current item is placed into the MVNET.READIMAGES file. Each read image is given a timestamp so that when the housekeeping process executes it examines the timestamps of all read images for this account and will delete any that are older than the age specified within the account profile.

The *Delete temporary session files* setting allows you to define when the temporary files for stateless connections are to be deleted. If an application passes an application GUID (session ID) into the connection request call, a dedicated temporary file will be created for that session ID so that information can be persisted for that session across invocation boundaries. mv.NET keeps track of when a session ID last contacted the server and if, when the housekeeping process executes, it finds that a session has not contacted the server for a period greater than that specified within the corresponding account profile the temporary session file will be deleted.

Other Account Profile Settings

The third tab on the account profile maintenance window allows you to define various miscellaneous account profile settings, as shown below:

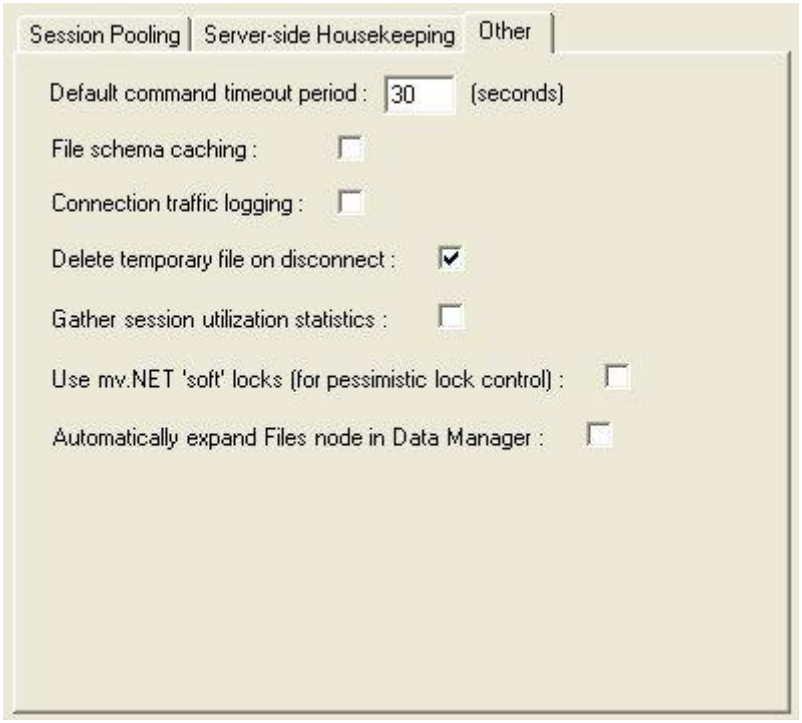


Diagram 5h : Other Account Profile Settings

The Default command timeout period field allows you to specify the default length of time (in seconds) that an application will wait for a response from the server after issuing a request. This may be overridden programmatically using the mvAccount.CommandTimeoutperiod property.

The *File schema caching* checkbox allows you to indicate whether you wish file schema data to be cached by the client to reduce round-trips to the server. This option should only be activated if the schema is not being actively developed/changed on a regular basis. Note, this option will only come into effect when new sessions are established. Also note that the Session Monitor window allows you to manually clear the schema cache without terminating a session.

The *Connection traffic logging* checkbox allows you to activate the logging of all messages passing between client and server for any sessions established using this account profile. Session logs are created in the following folder:

C:\Documents and Settings\All Users\Application Data\BlueFinity\mv.NET\Version4.0\Message Area\Live\Connections

Note, on Vista, Windows7 and Server2008 systems **C:\Documents and Settings\All Users\Application Data** is known as **C:\ProgramData**

Within this folder will be one folder per session log. The folder name will reflect the session id, the profile names used and also the date and time the log was produced. Within each session log folder will be multiple files holding a record of all message traffic.

The *Delete temporary file on disconnect* checkbox allows you to indicate whether the port-based temporary file automatically created by mv.NET at the start of a session (if it does not already exist) is to be deleted when a session terminates.

The *Gather session utilization statistics* checkbox allows you to indicate whether the Session Manager should record statistical information relating to the usage of sessions connected using this profile. This statistical information can then be viewed using the Statistical Analysis tool installed as part of the CIDSetup routine - a shortcut to which is created on mv.NET's Windows Start menu.

The *Use mv.NET 'soft' locks* checkbox allows you to indicate whether soft locking should be used for all clients connecting via mv.NET to this account. When soft locking is activated it results in mv.NET operating the pessimistic item lock table for an application as opposed to the O/S. It is typically used in situations where Multiple Session Sharing has been activated for an account profile which will, by

definition, result in multiple clients potentially sharing the same physical database process (PIB); thus rendering traditional READU item locking as unusable. It should be noted that soft locking should only be used when mv.NET is the only method being used to lock data. It will not ensure lock integrity with other applications using traditional READU item locking.

Please refer to the mv.NET Soft Locks section within the Data Manager chapter of this Guide for more information on soft locks.

The *Automatically expand Files node in Data Manager* checkbox allows you to indicate that when you connect into an account within the mv.NET Data Manager utility, the list of available files is to be automatically assembled and displayed. You should only tick this box if you are sure that assembling the list of files will not be a time-consuming process.

Creating a Login Profile

A login profile allows you to assign a logical name to a server and account profile pairing. The main reason for providing this capability is to encourage developers to avoid hard-coding the names of server and account profiles into their application. Therefore, you should use login profiles as the means of identifying the database you wish to connect to whenever possible.

To create a new login profile, select the Add Login Profile option from the Logins context menu. This option will prompt you for a profile name. After entering a name, a window will be displayed allowing you to select the relevant server and account profile names.

An alternative form of the login profile is one which is linked to a gateway profile. This allows logins to be processed by a remote gateway system. Please refer to the Gateways chapter for more information on this topic.

The Server Console Window

As well as providing a basic terminal emulation facility, the server console window allows you to perform several other important tasks:

- Server component download
- Demo SOP account download
- Account enabling

The server component download and account enabling actions *must* be performed to complete the setup of Core Objects. The download of the demo SOP account is optional.

Opening a Server Console Window

Under each server profile is a node called 'Server Console Window'. Double-clicking this node will result in a window allowing the address and port of the telnet listener for that server to be entered. Upon clicking OK in this window, a Server Console window will be displayed. There is also a button within the server profile maintenance window called 'Open Server Console Window' – clicking this button will also result in a Server Console window being displayed (using the address and port entered within the server profile definition).

Terminal Emulation

The black area within the Server Console window represents a terminal emulation region. The only emulation mode supported by the Server Console is VT220.

Using the emulation window, you may logon to the server and perform any of the command level activities supported by the MultiValued platform.

Server Component Download

Before a client application can perform any form of communication with the MultiValued server, you must download the mv.NET server components onto your database system. This, basically, involves the download, compile and catalog of 60 or so DataBASIC routines onto the server, along with the creation of a handful of control files. To do this, you will need to follow the following series of steps:

1. Using either the Server Console window or another terminal connection, create a new account on the system called 'MV.NET' or similar. This account will be referred to hereon as the 'mv.NET account'.

2. Within the Server Console window, logon to the mv.NET account and make sure that you are at the command prompt.

3. Select the console window's menu option: Action\Download Server Components. A extra region will be displayed at the foot of the window containing the following check boxes:

   - Force use of $INCLUDEs – this indicates that within the source code downloaded to the server, a '$' symbol will be prefixed to any INCLUDE statements. This is only needed for a small number of legacy MultiValued platforms.
   - Suppress use of named common – this prevents the mv.NET common area from being 'named'. You should only use this option after consulting BlueFinity.
   - Use extended delay after compile action – use this option if your MultiValued server takes a prolonged period (> 2 seconds) to compile/catalog programs. If the download process freezes part way, it is likely that you will need to use this option.

   After ticking the required options, click the Start button to commence the download process. You will be asked to confirm the database type.

At the end of the download process Windows Notepad will be invoked to display a trace of the download action. This trace will have been saved to disk, so you may close this window.

Account Enabling

Once you have downloaded the server components into the mv.NET account, you need to 'enable' each of the data/application accounts that you wish to access via mv.NET. This 'enabling' action simply refers to the creation of a handful of file (or 'Q') pointers with the data account back to the mv.NET account, and the cataloging of the downloaded DataBASIC routines (if necessary).

To enable an account, navigate your way to command level within the account and then select the Server Console's menu option: Enable Application Account. The option will ask you confirm the database type and that you are at command level within the account and will then perform a series of actions resulting in the execution of a program called 'MVNET.ENABLE', which check to make sure that the account is ready for mv.NET usage. A message saying 'Account structure OK' should be displayed at the end of its execution.

SOP Demo Account Download

To run some of the sample applications provided with Core Objects, you will need to download a demo data account onto your MultiValued server. Before doing this, you will need to create a new (empty) account on your MultiValued server called 'SOP', or similar. This will be referred to hereon as the 'SOP account'.

Within the Server Console window, logon to the SOP account and make sure that you are at the command prompt. Select the console window's menu option: Action\Download Demo SOP Account. Click the Start button to begin the download.

At the end of the download, a data generator program will be invoked on the MultiValued server within the SOP account. You need to complete the on-screen prompts in order to generate data within the files that the download process will have created.

Testing a Connection

Once you have created server and account profiles and also downloaded the server components onto your MultiValued server, you are ready to test whether the Data Manager (and hence you own application) is able to successfully connect into the account.

To run a connection test right-click an account profile node with the Data Manager explorer tree and select the Test Connection menu option. A Connection Test window will then be displayed, together with a series of trace messages.

A successful connection will results in the message 'AppGUID=[...] Port=[...]' being displayed at the foot of the test window.

Close the connection test window when you have finished.

If you experience difficulties getting the connection test to work, please contact your mv.NET support representative.

Connecting into an Account

After successfully testing your connection you are then ready to open a session within the Data Manager to view and possibly amend details within that account.

To open a database session double-click an account profile node within the explorer tree. This will expand the account node and present a list of sub-nodes as follows:

Master Dictionary	allows access to the contents of the MD/VOC of the account
Files	allows the list of files within the account to be viewed, along with subsequent access to these files
Queries	allows access to the predefined queries within the account. These queries are defined using the Data Manager

Viewing the List of Available Files

The first time that you expand an account profile's *Files* node you will presented with the following window:

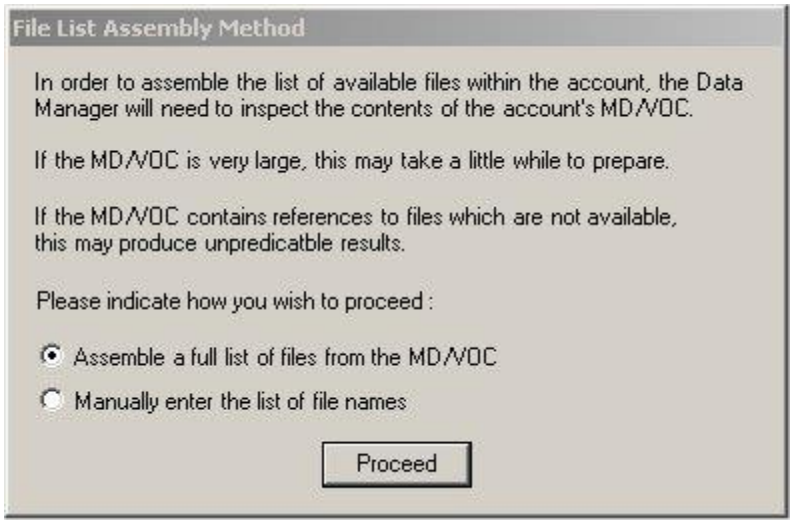


Diagram 5i : File List Assembly Confirmation Window

This window allows you to specify how the Data Manager will assemble the list of file names. The first option will result in mv.NET scanning the MD/VOC of the

account to try and deduce which entries refer to files (and will also scan the dictionary of all such files to try and determine what data levels are present). This is clearly a potentially time-consuming task and you should only select this option if:

a) The MD/VOC size is not excessive

b) The MD/VOC doesn't contain entries to files which are not accessible and will thus, possibly, cause the server process to crash. An example of this might be the presence of remote file pointers to a system which is not available.

The second option allows you to manually enter the names of the files that you wish to have listed within the explorer *Files* node. If you select this option, you will be presented with the following screen:

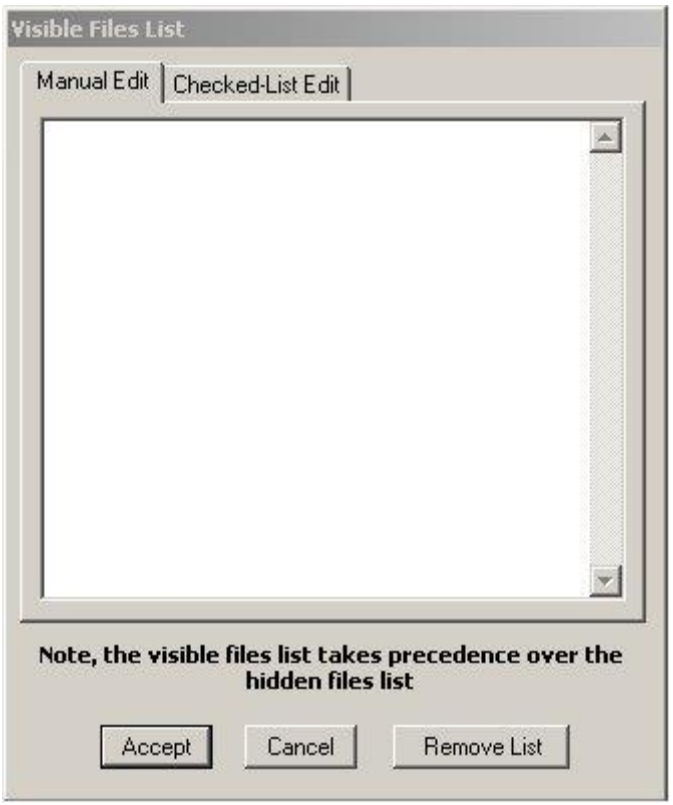


Diagram 5j : Files List Entry Window

The first tab on the above form contains a text box into which you can enter file names exactly as needed to open the files within DataBASIC. The second tab contains a checked ListBox which allows you to select which file names you require, however, to populate this checked ListBox with file names, mv.NET,

obviously, must scan the MD/VOC, therefore, it will present a prompt asking you to confirm that you wish to do this.

On clicking the *Accept* button, the file names that you enter/select within this form will be saved in a special item within the account's MD/VOC with an item ID of '{MVNETFILESVISIBLE}'. This is called the *Visible Files List* (VFL).

Whenever Core Objects is asked to provide the list of files within an account, the first thing that it checks for is the presence of the VFL item in the account's MD/VOC. If it finds the VFL item, it assumes that it contains the complete list of file names and returns this as the final result. If it doesn't find the VFL item, it will scan the MD/VOC for files.

It is important to realize that the VFL item does NOT contain the list of files that may be accessed from with your application – your application can open and access **any** file that you can access from command level within the account. What is does contain is the list of files names that will be returned by the FileList method of the mvAccount class – this is the method which the Data Manager uses to obtain the file list for an account. Please refer to the Class Library Overview chapter and the on-line help for more details on the FileList method of the mvAccount class.

If you initially manually enter a list of files names you may edit this list at a future point by right-clicking the *Files* node with the explorer tree and selecting the *Maintain Visible Files List* option.

Core Objects also has the concept of a *Hidden Files List* (HFL), which, if the VFL is not present, will be used to exclude names found during the MD/VOC scanning process. The VFL takes precedence over the HFL.

Accessing Alternative Dictionaries

If you have a situation where several files utilize a shared dictionary, you need to use the following syntax within the visible file list:

```
{dictfile} {datafile}
```

Where `{dictfile}` is the name of the file whose dictionary portion is to be used and `{datafile}` is the name of the file whose data portion is required (note the space separating the 2 names).

Note, if necessary, both `{dictfile}` and `{datafile}` can point to a specific data portion of a file by using the syntax:

```
{name},{data}
```

Where `{name}` is the name of the file and `{data}` is the name of the specific data portion.

Working with Files

Once you have a series of file names listed within the explorer tree you can perform a variety of tasks on these files. All these possible tasks are either listed on the right-click context menu for a file name or on the context menu for a data portion name within the file. You may view the list of data portions for a file by expanding the relevant file name node within the explorer tree. The following sections cover the more complex of these context menu options.

The Data Manager Toolbar

The Data Manager supports a toolbar that is used throughout the product. In the standalone version of the Data Manager the toolbar is at the top of the MDI form; in the addin version of the Data Manager the toolbar sits in the toolbar area of Visual Studio.

The functions available via the toolbar are as follows:

 Save Update	Saves the current amended details.
 Save As	Saves the current details under an alternative name/ID.
 Cancel Update	Cancels the current update action, losing any amendments made since the initial read or last save.
 Select Items	Allows a selective list of items to be assembled. This button is only available within the Item Maintenance form.
 Count Items	Allows a selective item count to be performed. This button is only available within the Item Maintenance form.
 Process Items	Allows multiple items to be processed (updated) in a common manner. This button is only available within the

	Item Maintenance form.
 Format	Allows the current program item's code to be formatted (indented). This button is only available when editing program code within the Item Maintenance form.
 Compile/Catalog	Allows the current program to be compiled and cataloged. This button is only available when editing program code within the Item Maintenance form.
 Run Query	Allows the query currently being maintained to be executed.
 Resize All Rows	Forces the height of all rows within the Item Maintenance window's grid to be reset back to the default height. This button is only available within the grid-style view of the Item Maintenance form.
 Select Dictionary Fields	Allows the list of dictionary fields which are to be displayed within the Item Maintenance window grid to be specified. This button is only available within the grid-style view of the Item Maintenance form.
 Show/Hide Dictionary Columns	Allows the display of dictionary fields within the Item Maintenance window's grid to be turned on and off. This button is only available within the grid-style view of the Item Maintenance form.

Maintaining the Schema of a File

The *Maintain File Schema* option from a file's explorer tree context menu displays the Data Manager's schema maintenance window. This form allows you to maintain the contents of a file's dictionary – both the native dictionary items and the mv.NET's *extended definition* items. For further information on extended dictionary definitions please refer to the chapter dedicated to this topic.

The Data Manager file node's right-click context menu contains a "Generate Extended Dictionary" option which can be used to automatically generate extended dictionary items based on the native dictionary definitions. You can then use the schema maintenance window to fine-tune these initial definitions as necessary.

The schema maintenance window presents a grid listing all of the dictionary names recognized within the dictionary of a file. Below the grid is shown both the native and extended definitions for the currently selected dictionary name.

The Extended Dictionary Definitions chapter and the mvSchemaItem section within the Class Overview chapter cover the content of extended dictionary definitions in detail.

Within this maintenance form you may add, amend or delete schema items as required.

Schema Item Visibility

Within the main grid of the schema maintenance window there is column titled 'Visible' – this column allows you to indicate whether a specific dictionary field is to be included in the list of schema definitions returned when the schema of a file is loaded within Core Objects. Specifically, it allows you to control the contents of the '{MVNETDICT}' item within the dictionary of the file. The paragraphs below explain this process.

When the FileOpen method of the mvAccount class is used you may specify that the schema of the file is to be retrieved. Alternatively, the Load method of mvFile.Schema (this returns a reference to an mvSchema object) can be used to load the schema at a later stage.

When a schema is loaded mv.NET first looks in the dictionary of the file to see whether an '{MVNETDICT}' item exists. If it does, it will only retrieve the names listed in this item, which thus means that the content of an mvFile.Schema will be (potentially) restricted to a subset of items. This is useful if you wish to restrict the dictionary names that are presented to the end user.

If the '{MVNETDICT}' item does not exists, all dictionary items within the file will be represented in the returned schema.

If you are using the MVNET.SCHEMA file to store extended dictionary items, the '{MVNETDICT}' item will be stored in this file as opposed to the dictionary if the file. See section on storing extended dictionary items for more information on this topic.

File Properties

The *Properties* option from a file's explorer tree context menu displays the Data Manager's General Properties maintenance window. This form allows you to maintain a range of mv.NET specific properties for the file as whole.

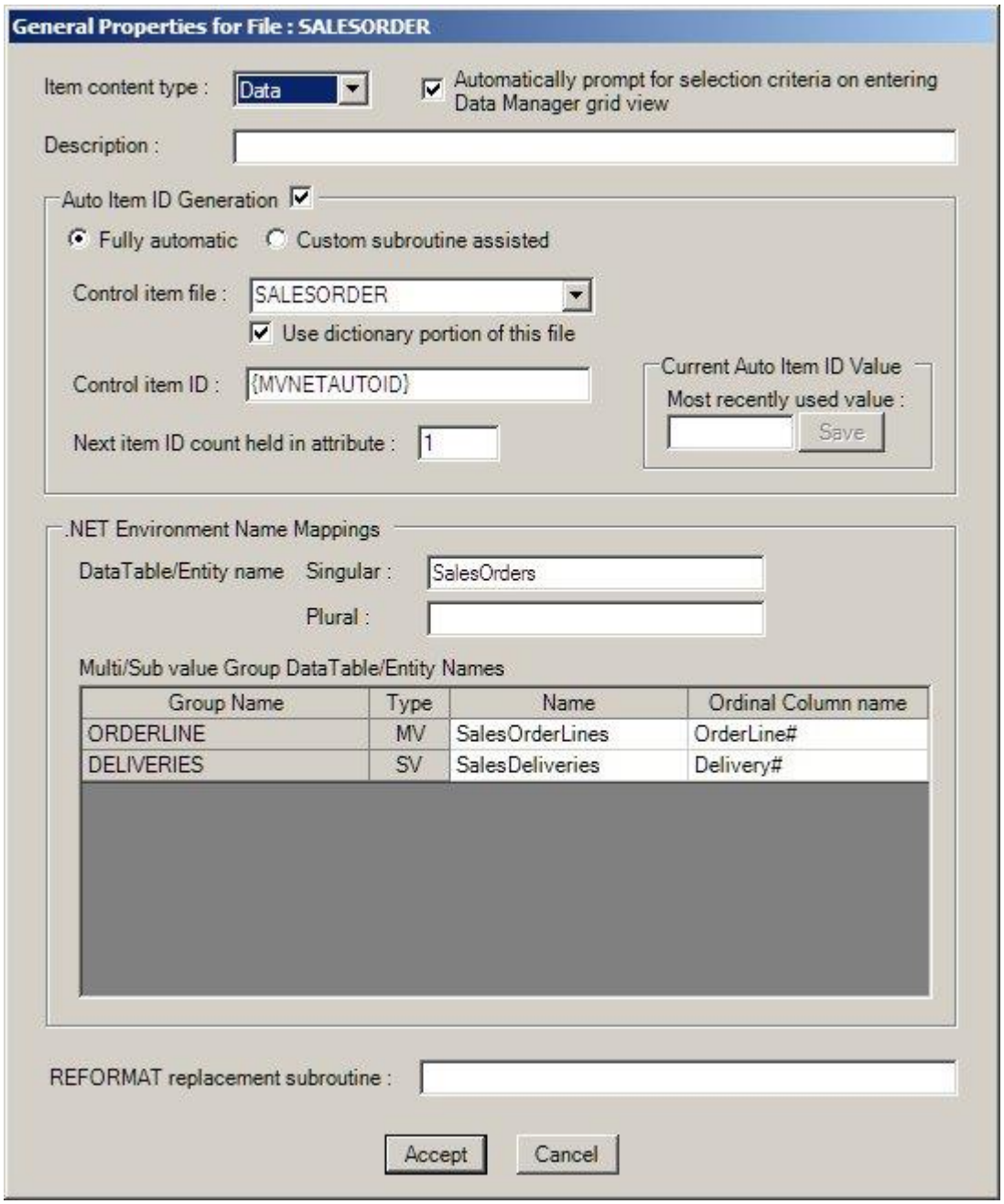


Diagram 5k : The File General Properties Window

The *Item content type* field allows you to specify the type of data held within the file – Data, Dictionary or Programs.

The *Automatically prompt...* checkbox allows you to indicate whether you will be automatically prompted for selection criteria when you choose to enter the grid-based data view within the Data Manager's Item Maintenance form. If ticked, this will prevent the grid view from automatically selected all items from the file - which do not want to do if the file contains a very large number of items (greater than 20,000).

The *Description* field allows you to enter a description of the purpose/content of the data held within the file.

The *Auto Item ID Generation* area allows you to specify whether/how automatic numeric item ID generation for new items within the file is to be performed.

There are 2 options here. "Fully automatic" will result in mv.NET generating unique numeric IDs for new items. In this scenario the item indicated by the file name, dictionary usage and item ID fields within the Properties form is automatically used to store the last number issued. The *attribute* prompt allows a specific attribute position to be used if you wish to store multiple auto ID counts in the same item. The current value of the auto item ID can also be viewed and amended using the edit field in the bottom left of the Auto ID area. If you edit the current value, you need to click either the "Save" button alongside the edit field (to save the value without leaving the Properties window) or the overall Accept button of the form.

If "Custom subroutine assisted" is selected, then a subroutine which you create will be called to supply new item IDs. This subroutine must conform to a pre-defined calling signature as documented below:

```
SUBROUTINE subroutinename (FILENAME, ITEMDATA, NEWID, ERRORMESSAGE)
```

Where:

FILENAME	the name of file to contain the new item. This allows a single subroutine to service new ID requests for multiple files if desired.
ITEMDATA	the data relating to the new item (if any is available at the time).
NEWID	needs to be assigned the value to be used as the new item's ID.
ERRORMESSAGE	needs to be assigned a value describing any errors that are encountered during the execution of the subroutine.

The .NET Environment Name Mappings area on the Properties form allows you to specify the names (singular and plural) by which the file will be known within the .NET environment and also the names of the normalized tables that will be

produced as a result of the Adapter Objects dynamic normalization process. This name is used by mv.NET's Adapter and Solution Objects component sets.

Each MultiValued and sub-value group defined within the mv.NET schema for the file is represented by a row within the Group Data Names grid. This information is used by mv.NET's Adapter and Solution Objects component sets. For Adapter Objects, you can define the Table name of the ADO.NET DataTable that will be created to hold each MultiValued/subvalued group data. You are also able to define the name of the column which is created within the DataTable to hold the ordinal MultiValued/subvalue position of each individual nested data element. For Solution Objects, you can specify the default Entity name to be used to represent the file along with the entity names to be used for the entities representing the multi/sub value grouping within the file. Please refer to the respective developer guides for more information on these topics.

The *REFORMAT replacement subroutine* input field allows you to specify the name of a user-written subroutine that will be called by mv.NET to provide data which would normally be generated via a REFORMAT command. You will need to use this feature if the REFORMAT command is not available for use on a particular file, an example of this being distributed files in UniVerse.

This REFORMAT replacement subroutine (RRS) must have the following declaration:

```
SUBROUTINE REFORMAT_DATA_FILEXYZ (ITEMID, DATAITEM, DICTNAMES, RETURNDATA)
```

Where:

`ITEMID` is the ID of a selected item.
`DATAITEM` is the data content of the above item.
`DICTNAMES` is a space delimited list of fields whose values are required.
`RETURNDATA` is an AM delimited list of values corresponding to the list of fields names in the DICTNAMES argument.

If you use this feature, you will need to implement data generation (within the RRS) for any field which is referenced via the DictionaryList property/argument within CoreObjects or via the Fields segment within the CommandText property of an Adapter Objects mvCommand instance.

Note, the values generated by the RRS should be in output format, i.e. the format that would normally be generated by the corresponding dictionary item.

You could obviously use a named common area to optimize file opening etc as required.

Maintaining Item Data

Upon expanding a file name within the explorer tree, a set of sub-nodes will be displayed. The first will be the *Dictionary* entry which allows you to maintain the dictionary items within the file.

The last sub-node will be the Queries node which allows you maintain queries associated with the file – see following Maintaining Queries section.

All the in-between nodes represent a data portion of the file.

The *Maintain Item* option from the data portion explorer tree context menu displays the Data Manager's Item Maintenance window.

Upon entering the item maintenance window an initial selection of the first 200 items within the file is performed and the IDs of the items are listed within the left-hand grid. To the right of the item ID grid is an editor text box which allows the content of the currently highlighted ID to the viewed/edited.

At the bottom right of the item maintenance window is a ComboBox allowing you to select an editor type. The value initially selected here is set according to the data content type setting within the file's properties; you only need to change this if you require editor options that are not available with the current settings – e.g. program compile and catalog.

At the bottom left of the item maintenance window are 2 buttons allowing you to choose which data view style you would like to use. The *Editors* style is the initial selection, displaying one item at a time. Alternatively, the *Grid* view displays data in a grid, showing 1 item per row.

Editors-style Data View

If you click the *Editors* button at the foot of the item maintenance window (which is the default style) you will be able to view/edit item data one item at a time on the right of the form. If you are using the Data editor (this can be selected using the Editor style ComboBox at the bottom right of the window) you will be able to select a second *Dictionary Derived* tab at the foot of the window which will allow you to view/edit data based on dictionary names. The *Refresh* button on this tab will force the schema for the file to be reloaded by the client and the item data to be re-read and redisplayed within the tab. The *Select Dictionary Fields* toolbar button may be used to specify which dictionary fields you wish to be included in this tab.

On entering the item maintenance form, you will be initially presented with the *Editors* style view. At this point only the first 200 items of data will have been selected for the file. On scrolling down through these items more data will be retrieved as necessary until the end of file is reached. Using this method, the Data Manager is able to safely open a file and select data irrespective of the number of items within the file.

Grid-style Data View

If you click the *Grid* button at the foot of the item maintenance window you will be presented with a grid showing one row per item - this is a read/write grid. You may manually adjust the height of individual rows (by dragging row divider lines in the left most column) to view more than one line of data - which you may wish to do if you are viewing MultiValued data as each multivalue occupies its own line within a row.

The *Show/Hide Dictionary Columns* toolbar button can be used to split the grid vertically to allow you to view data based on dictionary-derived value as opposed to the default (on the right) view of the physical content of items. The *Select Dictionary Fields* toolbar button may be used to specify which dictionary fields you wish to be included in the dictionary derived section of the grid.

The height of the header row within the grid may be manually adjusted (by dragging the row divider line under the header row in the left most column) to view extra definition data. The format of the header data for the physical data content section of the grid is as follows:

Line 1 : Column title (if available)
Line 2 : Attribute position
Line 3 : The name of the dictionary item which has been used to interpret the attribute position
Line 4 : The data type
Line 5 : The multivalue types

The format of the header data for the dictionary derived section of the grid is the same except that line 2 (attribute position) is omitted.

Unlike the *Editors* style view, the grid view will, by default, select all items from the file. If you wish to have option of selecting a subset of items from the file before entering the grid view (which we recommend if a file contains a very large number of items) you may use the setting within the *File Properties* window to force this.

Special Keystrokes

Within the Item Maintenance form the keystrokes Ctrl-] or Ctrl-2 will insert a multivalue mark; the keystrokes Ctrl-\ or Ctrl-3 will insert a subvalue mark. Within the grid view Ctrl-Return will insert a new multivalue position (i.e. add a new row) within a cell.

Queries

One of the nodes listed beneath a file node is the *Queries* node. This allows you maintain queries associated with the file. An alternative way of accessing queries is via the *Queries* node listed underneath an account profile node - this allows you to view the full list of queries for all files within an account.

Query Overview

The Data Manager allows you to create predefined queries which you may then incorporate into your rich client (WinForm) application. This feature can dramatically reduce the time taken to incorporate reporting and ad-hoc querying functionality into your application.

Please refer to the mvQuery classes within the Class Library Overview chapter for more details on the classes used to present query content at runtime. Also, please refer to the Binding Objects guide for details on the mvQuery control which is used to run queries within WinForm components.

Maintaining Queries

Upon double-clicking any query node within explorer tree, the Data Manager's Query Maintenance form is displayed. This form allows you to both define the query and to also view the results of executing the query.

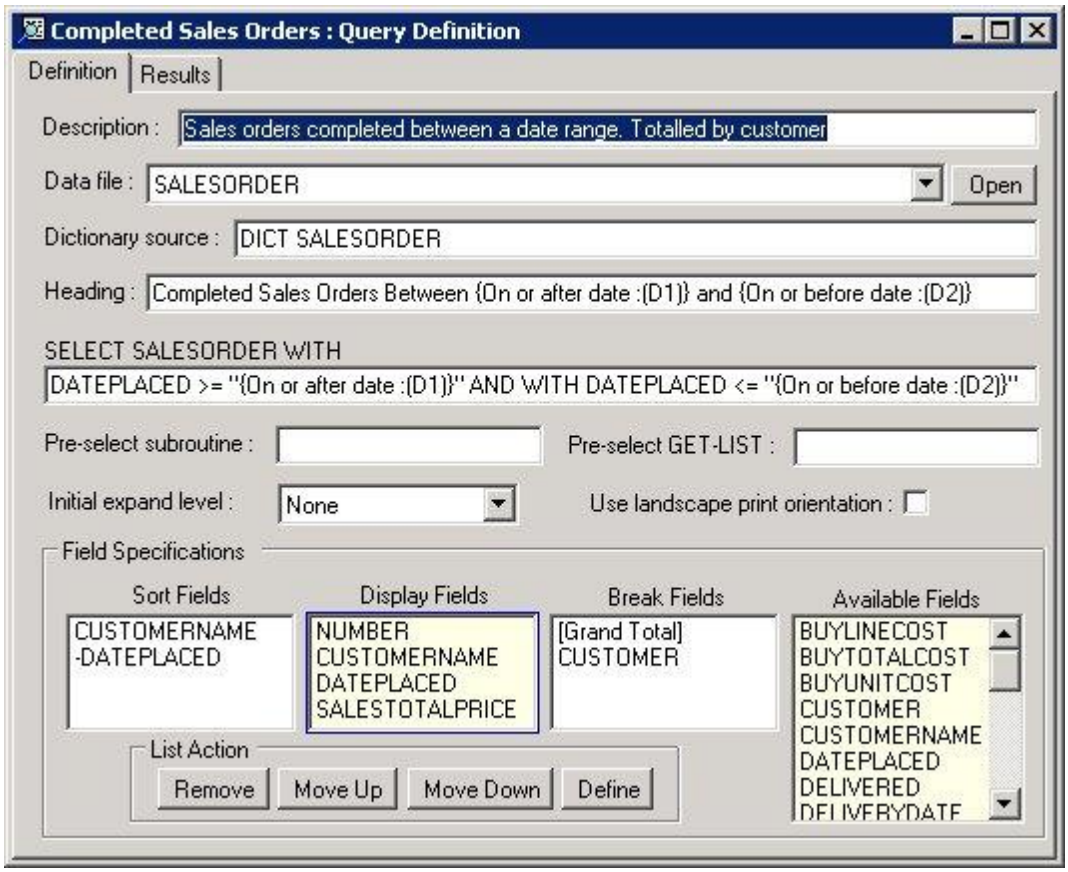


Diagram 5l : The Query Maintenance Window

Each field on the above form is explained below.

Description : Reference purpose only description of the query.

Data file : The file to provide data for the query. The *Open* button next to this field forces the *Available Fields* list at the bottom right of the form to be repopulated.

Dictionary source : The dictionary which will be used during the execution of the query. This defaults to the dictionary of the *Data file* but may be changed if necessary to point to an alternate dictionary.

Heading : The heading of the query. This may incorporate parameterized values as seen in the above screen shot. See Parameterized Values section below for more details on this topic.

Selection clause : This field allows you to enter the selection clause to be used to assemble the items to be included in the query results. If accepts standard selection MultiValued selection command syntax.

Pre-select subroutine : The name of the subroutine to call before the query is executed. This subroutine may serve a variety of purposes, the main one being

the creation of the Pre-select save-list which may be referenced by the **Pre-select GET-LIST** field - see below. The specified subroutine MUST have the following argument signature, i.e. it must support 3 arguments:

> **SUBROUTINE EXAMPLESUB** (QUERYNAME, ARGUMENTVALS, CANCELQUERY)
>
> QUERYNAME is passed into the subroutine and holds the name of the query being executed. ARGUMENTVALS holds an attribute mark separated list of (multivalue mark separated) runtime argument name/value pairs that have been collected just prior to the subroutine being called. CANCELQUERY can be set (to a value of '1') within the subroutine to force cancellation of query execution.

Pre-select GET-LIST : The name of the SAVE-LIST to be used to retrieve the list of item IDs against which to run to selection clause.

Initial expand level : Indicates how multivalues and subvalues are to expanded in the initial display of query results.

Use landscape print orientation : Indicates whether a landscape orientation is required is the query results are printed.

Field Specifications : The 3 field lists which may be specified can have entries added to them by clicking anywhere inside the target field list (so that the field list box shows a blue border) and then double-clicking the required entry I the Available Fields list.

Each entry in any of the field lists may have further definition information defined for it by highlighting the relevant name and them clicking the *Define* button – or by right-clicking any field list entry and selecting the Define option.

The ordering of entries in field lists may be amended by highlighting an entry and then clicking the *Up/Down* buttons

A field list entry may be removed by highlighting an entry and then clicking the *Remove* button.

Sort Fields : The sort order of the query results. The top most entry represents the highest (outer) sort order. A hyphen at the front of a field name designates descending sort order – the sort order of a field may be defined using the *Define* button.

Display Fields : The list of fields to display in the query results. The title, width and alignment of each column may be adjusted by highlighting an entry and then clicking the *Define* button.

Break Fields : The fields whose values are monitored for changes so that subtotaling and other change of value calculations may be performed. Note, only field names that appear in the Sort Fields list may be entered into this list. The

definition of a break field may be entered by highlighting an entry and then clicking the *Define* button, at which point the following window will be displayed:

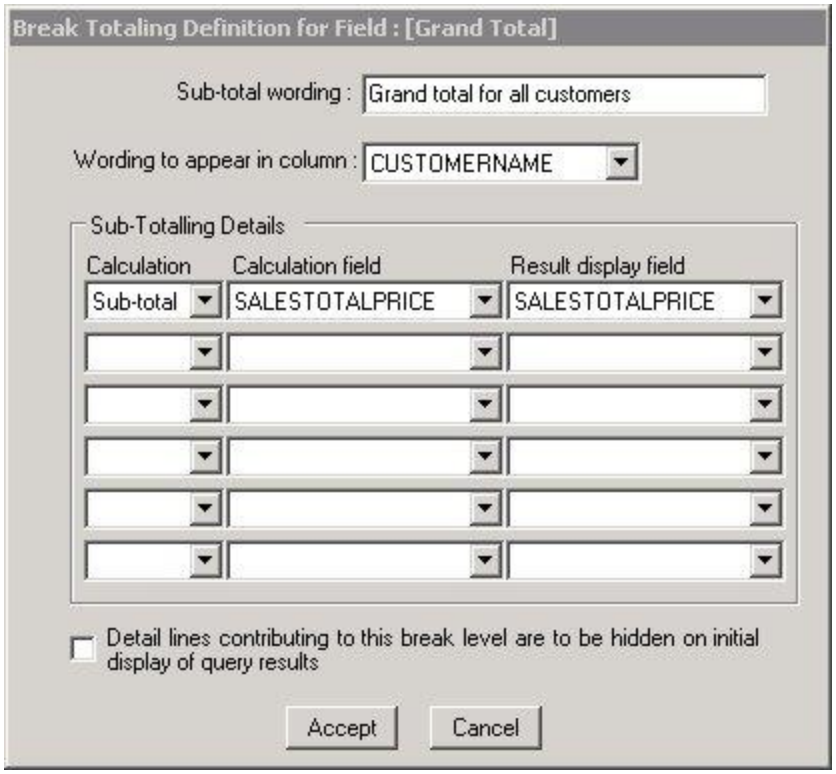


Diagram 5m: The Break Field Definition Window

In the above form, the wording to appear alongside the calculated value may be specified. If you wish to incorporate values from the last item within the break group, you may enclose dictionary names inside curly braces, e.g. `{NAME}` would be replaced by the value of the NAME field in the last item in the break group. The second input field on the form allows you to specify in which column within the query results the above wording text is to be displayed.

The *Sub-Totaling Details* area allows you to specify up to 6 separate calculations to be performed at this break level. The first column allows the types of calculations to be specified. The second column allows the field which is to be used to calculate the value to be specified. The last column allows the column which is to be used to display the calculated value to be specified.

The checkbox at the foot of the form allows you to indicate whether the display of the result lines which form this break group are to be hidden on initial display of the query results.

Testing the Query

In order to test the definition of the query you may click the Run Query toolbar button. The *Results* tab is used to display the output. Note, the results tab simply uses an mvQuery control to display the query results.

Parameterized Values

The heading and selection definitions of a query may both contain markers representing values that are to be supplied at runtime. The markers should be enclosed within curly braces and will result in a popup window (the 'argument collection window') being shown at runtime to collect the required value. The prompt text used in the argument collection window is whatever text appears inside the curly braces.

Note, if you want to use the same parameterized value in the heading and selection definition you must ensure that the text inside the curly braces is identical for both instances – if you do this, mv.NET will recognize that you are referring to the same argument and will only prompt for its collection once.

If you wish to force a parameterized value to be of a certain data type, you may place a data type descriptor within round brackets at the very end of the curly braced marker. The following data type descriptors are supported:

(D) – date
(D1) – date, being the first (oldest) date in a pair of dates
(D2) – date, being the second (most recent) date in a pair of dates

Note, the mvQuery control supports a Validate Arguments event which allows you to either force re-entry of argument values or cancel the execution of the query. Please refer to the Binding Objects guide for details on the mvQuery control.

mv.NET Soft Locks

The right-click context menu of an account profile node (and a login profile node) contain an option called "Maintain Soft Locks". This option is only enabled when an account is connected within the Data Manager.

Soft locks allow pessimistic item locking to be emulated when either:

a) Session sharing mode of "None" is being used and pessimistic item locks need to be carried across session login/logout boundaries. That is, an application is liable to use different sessions/database connections (PIBS) to service each individual session acquire/release action.

b) Session sharing mode of "Multiple session" is being used. In this scenario, several different clients may utilize the same session (PIB), in which case the standard READU item locking mechanism of the database engine is rendered ineffective. Additionally, as with a Session sharing setting of "None", clients may use different sessions (PIBS) to service each individual database request.

Soft locks are held within the MVNET.CONTROL file, one lock per item. The item ID of each of these soft lock items is as follows:

"LOCK *{accountname} {filename} {ItemID}*"

Where *{accountname}* is the name of the account holding the file, *{filename}* is the name of the file holding the item and *{ItemID}* is the ID of the locked item.

Note, mv.NET applications using soft locking will be "item lock incompatible" with other applications not utilizing mv.NET soft locking. This is because mv.NET soft locking does not utilize persistent READU locks. READU locks are only used for a fraction of a second whilst the MVNET.CONTROL file is being maintained by mv.NET.

The Maintain Soft Locks menu option displays a maintenance window which allows you to view the locks currently in force and remove any erroneous ones:

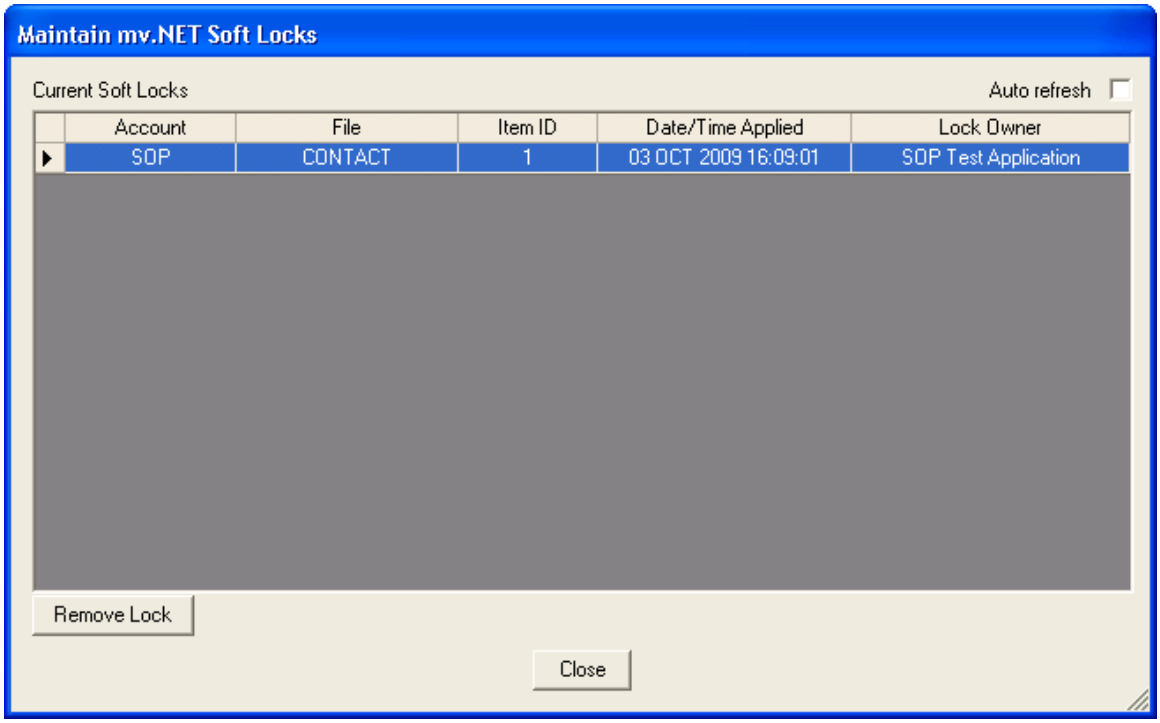


Diagram 5n: The Break Field Definition Window

The "Auto refresh" checkbox in the top right corner on this maintenance window will, when ticked, force the contents of the displayed soft lock table to be refreshed every second.

The "Remove Lock" button will remove the currently highlighted lock.

Extended Dictionary Definitions

This chapter covers the topic of mv.NET's extended dictionary definitions – an important topic for the developer especially if you intend to use mv.Net's Binding Objects or Adapter Objects products.

The Need for Extended Dictionary Definitions

All MultiValued databases support dictionary items, items which describe the data content of items within a file. However, the level of detail to which these 'native' dictionary items go down to is somewhat limited.

There are several places within mv.NET where a fuller, more complete definition of the schema of a file is required. For this reason, you are able to create 'extended' dictionary definitions to contain this extra schema data.

The extra definition data within extended definitions allows various aspects of mv.NET to better understand the structure and type of data within an item and also the relationships between attributes of data within the same item and also between items of data across different files.

Maintaining Extended Dictionary Definitions

The easiest way to maintain the extended dictionary items for a file is via the Data Manager utility. Please refer to the schema maintenance section within the Data Manager chapter for more details on this.

Alternatively, you may maintain extended dictionary definitions programmatically using the mvSchema and mvSchemaItem classes of the Core Objects class library. Please refer to the Class Library Overview chapter and the on-line help for more details on these classes.

Extended vs. Native Dictionary Definitions

It is important to note that the native dictionary definitions of a MultiValued database are still very important to mv.NET as these are used extensively to obtain information from the database. The extended dictionary definitions of mv.NET augment the native definitions and when the term 'schema' is used within any of mv.NET it is used to describe the combination of both the native and extended dictionary definitions of a file.

mv.NET is able to work with any of the native dictionary definition formats implemented by MultiValued databases.

The Storage of Native Dictionary Definitions

mv.NET stores its extended dictionary definitions in one of 2 locations; either in the dictionary portion of the relevant file or in a file named "MVNET.SCHEMA". mv.NET will automatically use the MVNET.SCHEMA file if it finds it within the application account, otherwise is will use the dictionary portion of files. There are 2 utility routines that can be used at command level within the database account to migrate extended dictionary items from the dictionary portion of files into MVNET.SCHEMA and vice-versa:

`MVNET.SCHEMA.MIGRATE`	-	moves extended items from dictionary portions of all files into MVNET.SCHEMA
`MVNET.SCHEMA.DEMIGRATE`	-	moves extended items from MVNET.SCHEMA into the relevant dictionary portion of files

If you wish to use the MVNET.SCHEMA method of extended items storage, you need to manually create the MVNET.SCHEMA file within the account and then use the `MVNET.SCHEMA.MIGRATE` utility. You must also create a visible files list using the Data Manager to use the MVNET.SCHEMA method of storage. mv.NET will create a separate data portion within the MVNET.SCHEMA file for each file being used via mv.NET as and when it is required to do so.

Where extended definitions have been created by the developer (note, extended definitions are never generated automatically by mv.NET) there is one extended definition item for each native dictionary item that has been 'extended'. The item ID of an extended definition item takes the form: **`{xyz}`** – where xyz is the name of the native dictionary item – i.e. extended dictionary items have the same item ID as the native item to which they relate but are enclosed within curly braces.

Extended Definition Fields

In all, there are around 30 separate extended dictionary definition fields which may be defined. You need to define whatever fields, for whatever native dictionary items are relevant to your application development requirements. You may do this preemptively all at once or you may do it on a piecemeal, as required basis.

The following sections cover all the available extended definition fields and also describe where each field is used by Core Objects, Binding Objects and Adapter Objects. The extended dictionary field name used as the title of each of the following sections is the same as the corresponding property name of the mvSchema class. Note, the number in brackets following the name of the extended field in the following sections represents the attribute number that the extended field data in stored in within the extended item.

AttrPos (1)

The attribute position of the field. If no extended definition exists for a dictionary item, this extended field will be set to the attribute position as indicated by the native dictionary item.

The AttrPos field is used to extract data based on name from within data items and is used throughout Core Objects for this purpose.

Title (2)

The descriptive title of the field. If no extended definition exists for a dictionary item, this extended field will be set to the Title text as indicated by the native dictionary item.

The Title field is used by Binding Objects when a field is referenced in columnar display/input formats, such as within grids or queries.

Justification (3)

The default display/input justification of the field. If no extended definition exists for a dictionary item, this extended field will be set to the justification setting as indicated by the native dictionary item.

This definition field is used in the same way as the Title field.

Width (4)

The default display/input width of the field. The unit of measure here is character count. The appropriate conversion of this to pixels is performed by Binding Objects automatically. If no extended definition exists for a dictionary item, this extended field will be set to the width value as indicated by the native dictionary item.

This definition field is used in the same way as the Title field.

DataType (5)

The fundamental data type of the field. This may be one of the following:

DataAlphaNumeric
DataNumeric1
DataNumeric2
DataNumeric3
DataNumeric4
DataNumeric5
DataNumeric6
DataNumeric7
DataNumeric8
DataNumeric9
DataInteger
DataNumericEmbedded
DataDate

DataTime
DataBoolean

The DataAlphaNumeric setting indicates a free text field. The DataNumeric*n* settings indicate a numeric field of *n* decimal places which is stored on the database with no decimal point. The DataNumericEmbedded setting indicates a numeric field with its decimal point stored within the database.

This definition field is used in numerous places by Core, Bindings and Adapter objects, mainly in the areas of data validation and the decision of which visual controls to use in the representation of data on forms.

MVType (6)

This field indicates whether the field is MultiValued or subvalued. It may be set to one of the following values:

Singular
Multivalued
Subvalued

In Binding Objects, this definition field is used to determine the visual control that will be used to show data on a form. Both Binding and Adapter Object use this field to control relationships between data.

MVGroup (7)

The name of the multivalue grouping to which this field belongs. This is logical name that can be decided by the developer, but, all fields that belong to the same group must have their MVGroup set to the same name.

Core, Binding and Adapter Objects all use this field to automatically keep all multivalue groupings in-step when multivalue positions are added or deleted. Adapter Objects uses this definition to control its Dynamic Normalization process - see Adapter Object guide for more details.

SVGroup (8)

The name of the subvalue grouping which this field belongs to - see MVGroup.

InputMandatory (9)

Indicates whether the input of this field is mandatory. This is used by Binding Objects' input validation process.

InputPrompt (10)

The default prompt text for a field. This is used by Binding Objects' databound control creation wizard.

InputDefault (11)

The default value for this field. This is used by Binding Objects' to initial populate a databound control with data when new items are created.

InputMin (12)

The minimum valid value for a field. This is used by Binding Objects' input validation process.

InputMax (13)

The maximum valid value for a field. This is used by Binding Objects' input validation process.

InputCasing (14)

Indicates how the upper/lower casing of a field during the input process is to be controlled/forced. This is used by Binding Objects' input validation process.

InputOptions (15)

The list of valid options for a field. This can be used by Binding Objects to populate the options list of ComboBoxes and other controls.

InputInOptions (16)

This indicates whether a field's value has to be within the InputOptions list (see above). This is used by Binding Objects' input validation process.

BooleanTrue (19)

The string value that represents a Boolean value of 'True'. This field is only relevant for extended fields with a data type setting of 'Boolean'.

BooleanFalse (20)

The string value that represents a Boolean value of 'False'. This field is only relevant for extended fields with a data type setting of 'Boolean'.

Dependencies (21)

The fields that are used to derive/calculate the value of a field. This is used by both Core and Binding Objects to automatically detect when the value of a calculated field needs to be regenerated.

LinkedFile (22)

The name of the file to which a field is a foreign key. This is used by the Solution Objects entity interrelating process and by Adapter Objects in its AutoLinking of DataTables.

LinkedFileIDField (23)

The name of the item ID field within the LinkedFile.

LinkedFileDescField (24)

The name of the primary descriptive field within the LinkedFile.

Notes (25)

Allows general programmer/data usage notes to be stored with a definition. This field is for programmer reference only.

AdapterColumnName (26)

This is only used by Adapter Objects and represents the name of the DataColumn that is created to hold the value of a field within an ADO.NET DataTable. It is also used by Solution Objects as the default name of the entity property representing this field.

SysDelimTrans (27)

This setting allows system delimiter translation to be turned on for a particular data field. This feature is only relevant for calculated fields representing multivalued/subvalued data. On most MultiValued database platforms, calculated values have VM/SVM characters automatically replaced by space characters. This setting allows the developer to indicate how these space (or other) characters are to be converted back into VM/SVM characters.

SysDelimTransVM (28)

Allows the character within calculated field values that represents a value mark to be defined.

SysDelimTransSVM (29)

Allows the character within calculated field values that represents a subvalue mark to be defined.

LinkedFileProperty (30)

Allows the name to be used by Solution Objects when a property relating to the linked file is created.

CompoundDataSep1 (31)

The character separating individual elements of data within an attribute containing multiple data elements.

CompoundDataPos1 (32)

The index position if the required data element within an attribute that contains multiple data elements. If this value is blank, all data elements will be returned as a multivalued list.

CompoundDataSep2 (33)

The character separating individual sub-elements of data within an attribute containing multiple data sub-elements within each main data element.

CompoundDataPos2 (34)

The index position if the required data sub-element within an attribute that contains multiple data sub-elements. If this value is blank, all data sub-elements within the specified CompoundDataPos1 will be returned as a multivalued list.

Compound Data Handling

The Compound Data feature of the extended dictionary allows you to define how mv.NET is to break apart attributes that contain multiple data elements delimited by custom value separators. This breaking apart allows for both data retrieval of sections of data and update of individual parts.

An important thing to note for the compound data feature is that if you need to select data items based on the value of a compound data element you still need to create a native dictionary definition that performs the raw data extraction.

It's worth noting that if you only ever require read-only access to the separate elements of data, you don't need to use the compound data feature. You simply need to create a native dictionary definition that extracts the relevant data and you will be able to use this as a calculated field with mv.NET. However, calculated fields do not support write-back to the database. This is what the compound data feature provides.

The compound data feature works in conjunction with native dictionary items to provide mv.NET with enough understanding of the internal structure of the data to allow it to extract/add/insert/delete individual elements of data.

Let's illustrate this feature by the use of an example.

Say we have a file with a field (DATEPARTS) containing the following data:

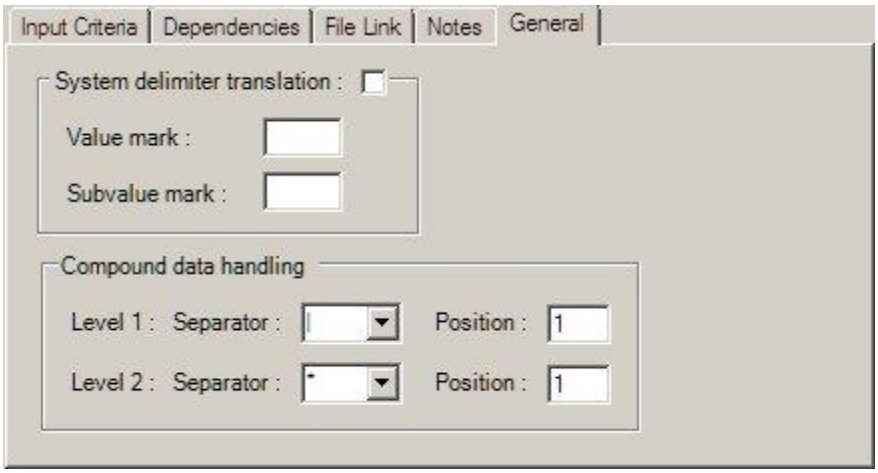
13634*AB76|13638*SG630

Here we have a list of data value pairs, with each pairing separated from the next by the use of a pipe ("|") character. In each pairing the 2 values are separated by the use of an asterisk character; the first value is a date (internal representation), the second value is a part number.

Let us suppose that we want to define the following fields:

1. A field that returns/allows update of the first date value (FIRSTDATE)
2. A field that returns/allows update of the list of part numbers (PARTNUMBERS)

For both fields we need to specify a compound data separator of "|" for the first level and "*" for the second level. This is done within the "General" tab of the "Extended" dictionary definition tab of the Data Manager's Schema Maintenance form:



Also, and very importantly, we need to specify in the Dependencies tab of the extended field definition the fact that this field is dependent upon the base (DATEPARTS) data field for its value.

Working with a Single Compound Data Value

For the first field (FIRSTDATE), we need to specify a value of 1 for both the level 1 and level 2 positions. We also, if we intend to use this field in selection statements to be run against the database, we will need to create a native dictionary item that performs the relevant extraction. Below is an example I-type dictionary that does the job:

```
001  I
002  FIELD(FIELD(DATEPARTS,"|",1),"*",1)
003  D2
004  First Date
005  12L
```

If you will never reference the field in database selection commands, the native dictionary for the FIRSTDATE field can simply be a copy of the DATEPARTS field, except for the data type, which needs to be set to Date.

OK – so, in summary, we have created a native dictionary item called FIRSTDATE. We have extended this dictionary item in the Data Manager's Schema Maintenance screen to:

a) specify a data type of Date

b) specify a dependency on the DATEPARTS base field in the Dependencies tab

c) specify the Compound Data settings in the General tab as per above

This then provides us with read/write access to the FIRSTDATE field.

Working with a List of Compound Data Values

For the second field (PARTNUMBERS), we need to specify a value of "2" for the level 2 compound data separator position – the level 1 position being left blank or set to "0". A blank/zero level 1 position indicates that we require a list of values returning.

As with the FIRSTDATE field, if we intend to use this field in selection statements to be run against the database, we will need to create a native dictionary item that performs the relevant extraction. Unless you are a dictionary item expression guru, this may well involve coding a custom basic subroutine to do the job and invoking this subroutine via the dictionary item.

In summary, we have created a native dictionary item called PARTNUMBERS. We have extended this dictionary item in the Data Manager's Schema Maintenance screen to:

a) specify a data type of Alphanumeric

b) specify a dependency on the DATEPARTS base field

c) specify the Compound Data settings as per above

This then provides us with read/write access to a multivalued list of part numbers via the PARTNUMBERS field.

The Session Manager

This chapter describes the purpose and function of mv.NET's Session Manager and session pooling components.

The Need for Session Management

Session Management (also referred to as Session Pooling) within mv.NET is simply the ability to maintain one or more (i.e. a pool of) open database connections across application requests for database access. That is, if an application requests a database session during its execution and then releases it upon termination the session may be kept open ready for the same or another application to utilize it at some future point in time without needing to perform a fresh database connection initialization phase.

The concept of a session pool may, thus, be defined as a dynamic, self adjusting collection of database connections able to expand and contract between defined bounds according to the demand for database access across a population of application clients.

The way in which session management works on a given system can be controlled to allow the precise requirements of an installation to be met. This is done by setting a number of control values within the product. See the Configuring Session Management section below.

In its creation of database connections, the Session Manager communicates with the License Manager in order to ensure that occurrence of database sessions complies with the purchased number of licensed connections. Please refer to the License Manager chapter for more details on database license management.

What is the Session Manager?

The Session Manager comprises 4 executables:

```
mvNET.SessionManager.Service.exe
mvNET.SessionManager.exe
mvNET.SessionCluster#n.exe
mvNET.CoreMonitor.exe
```

These 4 programs are responsible for overseeing all aspects of session management on a given system.

The mvNET.SessionManager.Service.exe (as its name implies) runs as Windows service. It is during the startup of this process that the other processes are launched.

The mvNET.SessionManager.exe process is a .NET remoting server which is responsible for managing session pooling. As part of its duties, this process will launch one or more processes which are to host the actual database connections, these processes are known as 'Session Clusters'. Each session cluster may host several database connections, the precise number being controlled by the Session Manager Settings – see Configuring Session Management below. Each cluster has a unique process ID, being of the form mvNET.SessionCluster#n.exe, where n represents the session cluster number.

The mvNET.CoreMonitor.exe process is responsible for providing monitoring services across all the session management executables.

The Location of the Session Manager

The Session Manager needs to be installed onto the system which will hold one or more pooled database sessions. To aid understanding of which system this should be, 2 scenarios are explored.

The first scenario is where software developer Bill wants to use Core Objects in the application he is developing. The MultiValued database that Bill is going to access requires a connection to be established via a Telnet IP link, and therefore will entail a connection negotiation sequence to be performed as part of the initial connection process. Negotiated connection sequences, such as this, can often take a second or two to complete, therefore Bill has, rightly, decided to use the Session Manager to hold open his database connections across application invocations. This means that when he runs his application to test something, the time taken to obtain a database connection will be practically instantaneous, apart, of course, from the very first time this is done.

Bill, therefore, needs to install the Session Manager on his own workstation, as he needs to, in effect, operate his own personal connection pool irrespective of any other users. If he installs the Client Interface Developer, this will be done automatically for him.

The second scenario is where company XYZ is developing a web-based application, incorporating Core Objects to access their MultiValued database to retrieve stock information. As a web-based application, their code is organized into atomic units that are executed on each HTML page invocation, typically first opening a database, then reading/writing data and finally terminating the connection. This is an ideal candidate for session management/pooling as the Session Manager can keep databases connections open across page invocations and therefore avoid each page having to establish a new database connection every time it is accessed.

In this situation, each web page is, in effect, an application client, and therefore the Session Manager should be installed onto the web server using the SRDK setup routine.

From the above scenarios in can be seen, therefore, that the Session Manager needs to be installed on the system which is hosting the processes that need to share a database session pool, be that an individual's workstation or a web server/service host.

Multiple Session Managers

For very large installations or for installations where there is a need to provide system redundancy in the role of session manager, it is possible to run mv.NET session management on more than one system. In such situations, each session manager operates independently, and it is up to either the client application or the networking infrastructure to control which session manager a client will access at any given point in time. Please refer to the License Manager chapter for details on how license management is organized in such scenarios.

Configuring Session Management

There are 2 places where the operation of the Session Manager can be controlled via settings, both accessed via the Data Manager application.

The first is the Session Manager Settings node within the main explorer treeview within the Data Manager. Upon opening this node, the following form will be displayed:

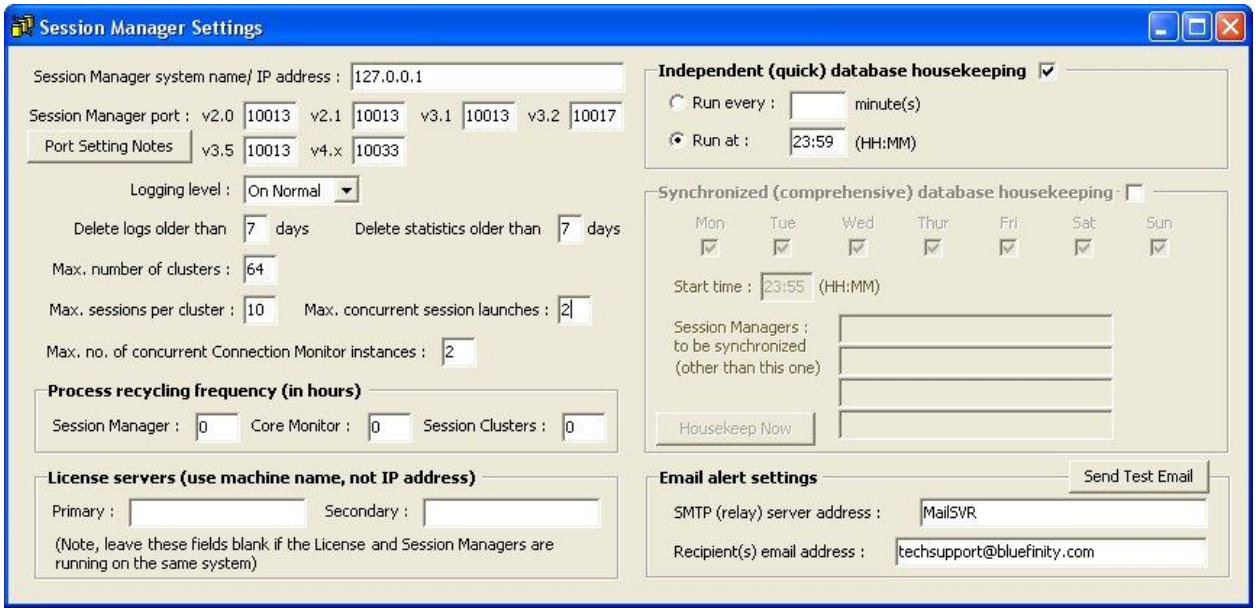


Diagram 7a : The Session Manager Settings Window

The various fields on this form are explained below:

Session Manager system name: This field holds the IP address or resolvable machine name of the system hosting the Session Manager. This will normally be the same as the host system but can be different if the Session Manager is running on a separate system.

Session Manager port: The port on which the session manager 'listens' for communications. Note, it is on this port that the mvNET.SessionManager.exe listens; the mvNET.CoreMonitor.exe listens on this port number plus 1. The mv.NET.SessionCluster.exe listens on this port number + 2 + the cluster number. The default value for port setting is 10013.

Logging level: Allows the internal messages flowing between the session management components to be logged. If this level is set to anything other than 'None', these messages may be viewed within the Session Monitor application.

Max. number of clusters: The maximum number of session cluster processes which may be concurrently active at any one time.

Max. sessions per cluster: The maximum number of sessions which may be hosted by a single cluster.

Max. concurrent session launches: Allows the maximum number of concurrent session launches within a cluster to be specified. This setting is used to prevent a new session launching cascade, whereby if many clients all simultaneously request a session, the instantiation of new sessions is choked to prevent an unnecessarily large number of new sessions being created.

Independent Database Housekeeping: During its normal course of execution, mv.NET generates a range a temporary data on the server. This input area allows you to indicate when the automatic housekeeping of this temporary data is to run.

Synchronized Database Housekeeping: If you wish to have all the database-resident mv.NET temporary data deleted, the mv.NET Session Manager service needs to perform a restart of itself. This input area allows you to specify when such a restart should be performed. Tick the days when service restart should be triggered and indicate at which time of day this is to be done. If you have more than one system running session management, only one should be set to run synchronized housekeeping. The system names of the other session managers can be specified within the lower section of this input area.

Email alert settings: If you have an email server available that is configured to allow the relaying of emails you may specify the relevant details in the 2 email alert input fields to receive emails when certain events are detected by the Session and License managers. If you require emails to be sent to multiple addresses you may enter more than one email address, each separated with a semi-colon in the recipients input field. The following events will trigger the transmission of an email to the specified email addresses:

- The detection of an invalid license by the License Manager
- A change of status of the Primary/Secondary License Managers

Please refer to the License Manager chapter for details on the purpose of the 2 License Server input fields.

The second place where settings controlling the operation of session management are held is within account profile definitions. Each account has its own series of settings that indicate how the pool of sessions for that specific account is to be operated.

Minimum number of sessions to pool: Allows you to indicate the minimum number of sessions that may be contained within the pool.

The **Launch minimum sessions on Session Manager service startup** checkbox allows you to indicate whether this minimum number of sessions should be automatically established upon Session Manager service startup, or whether it should be reached through the natural increase in demand for connections. However, once this minimum number of sessions is reached, the Session Manager will not allow them to be released.

Maximum number of sessions to pool: Allows you to specify the maximum number of sessions that may be held open (ready for allocation) within the session pool at any one time. The minimum allowable value in this field is 1.

Terminate pooled sessions after n minutes of existence: Allows you to indicate the maximum life span of a session within the pool. When a session has been in existence for greater than the amount of time indicated by this field it will be gracefully terminated. A value of zero here indicates that session's lifespan is unlimited (unless otherwise terminated)

Terminate pooled sessions after n seconds of inactivity: Allows you to indicate the maximum number of seconds that a pooled session may remain inactive within the session pool before it is automatically terminated. A value of zero here indicates that sessions are to be kept open indefinitely. A value of 1 indicates that as soon as a session becomes free it will be terminated - in effect preventing session pooling.

Free-up allocated sessions after n seconds of inactivity: Allows you to indicate the maximum number of seconds that a pooled session may remain in an 'Allocated' state with no activity. If a non-zero value is entered here, the status of an allocated session will be adjusted in accordance with the setting of the 'Free-up style' input field (see below). This field should be used with care. It is primarily provided to allow the session manager to automatically recover from situations where an application has terminated in a non-graceful manner, i.e. without performing a session disconnection action (mvAccount.Logout). A value of zero here indicates that no automatic session freeing is to be performed.

Free-up style: This input field allows you to select the style of free-up action invoked when the free-up period defined above expires. There are 2 options:

- Set status to "Free"
- Remove from session pool

The first option is designed to cater for client crashes; that is, situations where a client application has aborted abnormally without indicating to the session manager that it has finished with a session. When the free-up period expires, the state of the session is set to "free". Note, the assumption here is that the associated server process is still active and able to respond to further requests.

The second option is designed to cater for server-side crashes; that is, situations where the server-side process has fallen into a debug state or has otherwise 'died' without any notification having been sent back to the client-side session manager. In such a situation (when the free-up period expires), a setting of 'Remove from session pool' will result in the corresponding entry being removed cleanly from the session pool.

Session sharing mode: This dropdown list allows you to specify how pooled sessions are to be shared across multiple client processes/threads. The following options are available:

None - No session sharing will be allowed. That is, once a client process has been allocated a session for this account from the connection pool, it will have exclusive access to that session until it explicitly releases it.

Single - All processes requesting access to a session that is connected to this account will be 'channeled' through a single session. This option should be used only in a development environment where it is likely that processes will crash or will be manually halted prematurely before releasing a session.

Multiple - All processes requesting access to a session that is connected to this account will be allocated a free session for the duration of a single server request (round-trip) only. After a response is received back from the server, the session will be automatically placed back into the session pool and made available for other clients. It is important to note here that the client application will still 'think' that it has a permanent connection to the database - the session allocate/free activity happens automatically. Multiple session sharing allows a greater degree of connection sharing amongst clients but does place a slightly higher load on the Session Manager.

The Multiple session sharing option is typically used to allow multiple rich client applications to be multiplexed through a relatively small number of database connections. However, there are several restrictions in this setting:

1. O/S level pessimistic locking is not available. To work around this, mv.NET supports a concept of 'soft' locks, which essentially means that mv.NET operates the pessimistic item lock table for an application as opposed to the O/S. This

allows pessimistic locking to be used in a situation where several discrete clients share the same physical database process (PIB). It should be noted that soft locking should only be used when mv.NET is the only method being used to lock data. It will not ensure lock integrity with other applications using traditional READU item locking. Soft locking is activated on the "Other" tab of the account profile definition. Please refer to the mv.NET Soft Locks section within the Data Manager chapter of this Guide for more information on soft locks.

2. Index-based data access is not available. That is, the use of the mvFile.IndexSelect method is not allowed.

Auto release sessions: This field is only enabled when the session sharing mode is set to "Multiple" and allows you to specify (in milliseconds) the length of time that the automatically acquired database connection will be retained before being released. If you require the connection to be released as soon at the database round-trip is completed, this value should be set to zero. Using a non-zero value here allows you to reduce the session allocate/release burden on the Session Manager but at the cost of clients retaining connections for longer periods of time.

Queue for free session: This checkbox allows you to specify whether a process will wait for a session to become available if all current sessions are allocated and the maximum concurrent session limit of the session pool had been reached. If this option is not checked, an exception will be raised if the above set of circumstances arises.

The **Queuing Poll Interval** field allows you to define how frequently a process will poll for a free session if it has been placed in a wait queue. A process will poll for a free session n number of times (as indicated by the **Maximum number of poll attempts** setting) before an exception is raised.

The Session Monitor Application

To allow you to view and control the session pools on a system, a monitor program is supplied. This program is located in:

```
Program Files
(x86)\BlueFinity\mv.NET\Version4.x\bin\mvNET.SessionMonitor.exe
```

A short-cut to this program is also placed on the Start\Programs\mv.NET menu by the installation routine. Upon invoking this application, the following form is displayed:

Diagram 7b : The Session Monitor Application

There are 3 tabs contained within the monitor's form:

The **Active Sessions** tab lists all of the sessions and session clusters currently known to the Session Manager.

The **Message Logging** tab allows you to view the messages being generated by the Session Manager and other internal components of the system.

The **Database Licensing** tab allows you to view the Database Access Licenses (DALs) that are both installed (available) and currently in-use.

Within, the Active Sessions tab, there are right-click context menus which allow a range of maintenance operations to be performed.

The Message Logging tab allows you to view both the messages that have been generated by the current instance of the Session Manager and those generated by previous instances.

The menu bar of the Session Manager allows the cache of schema data for a given account profile to be manually cleared (see section Other Account Profile Settings). It also allows you to view the mv.NET database licensing information being used by the License Manager – see following chapter for more details on this topic.

At the foot of the Session Monitor window are 4 buttons which allows the status of the Session Manager and License Manager services to be controlled. The current status of the service is also shown. Note, these buttons are only shown if these services are running in the same system as that being used to run the Session Monitor utility.

Monitoring a Remote System

The Session Monitor application can monitor a Session Manager which is hosted on a different system to that on which the monitor is running. For the monitor to know where the Session Manager is running, the ConfigurationPath file must contain the address of the Session Manager – see section The Configuration Database for more details.

The License Manager

This chapter describes the purpose and function of mv.NET's License Manager, along with the basic principles of database access licensing within mv.NET. It also covers the procedure for applying for, installing and viewing database access licenses.

The Role of the License Manager

Each concurrent mv.NET session to a specific database instance consumes an mv.NET database access license. You may purchase the required number of licenses for each of the database instances to which you require access via mv.NET. Each database access license is registered with a specific License Manager; a Session Manager must have access to a License Manager to successfully establish database connections. The License Manager ensures that (as a maximum) only the purchased number of concurrent sessions to a specific server is in effect at any one time.

The License Manager Service

The License Manager manifests itself as a Windows service. This service will be automatically installed as part of the CID setup or SRDK setup installation routines.

Specifying the License Manager Address

The Session Manager needs to know where to find the License Manager. This can be done using the Session Manager Settings node within the Data Manager explorer treeview. This window allows both a primary and secondary address/system name to be supplied. If only one License Manager is in operation, only the primary address should be entered. If no addresses/names are entered, the Session Manager will assume that the License Manager can be found at localhost. The License Manager listens on the remoting port number used by the Session Manager minus one. See the following Multiple License Managers section for details on running multiple License Managers.

Licensing Principles

The basic concept of database access licensing within mv.NET is that an access license for a set number of concurrent connections to a specific database installation needs to be installed into a License Manager. In order for a Session Manager to establish a connection (session) to a database, it must be able to contact a License Manager that has the appropriate access license installed. The License Manager checks the number of active sessions vs. the number of licensed sessions in order to decide whether the Session Manager is to be allowed to establish a new connection.

Applying for Database Access Licenses

In order to receive a database access license, you first need to request it. This can be done using either the Data Manager (mvNET.DataManager.exe) or the License Request utility (mvNET.LicenseRequest.exe). The License Request utility is provided as a standalone utility and is intended for use by system integrators as a way of pre-requesting database access licenses in advance of a pending mv.NET installation.

NOTE, when applying for database access licenses, the Data Manager or License Request utility MUST be run on the system which is going to or which is currently running the License Manager service.

Using the Data Manager, you need to right-click the appropriate server profile and select the 'Request Database Access License' option. On doing this, the following window will be displayed:

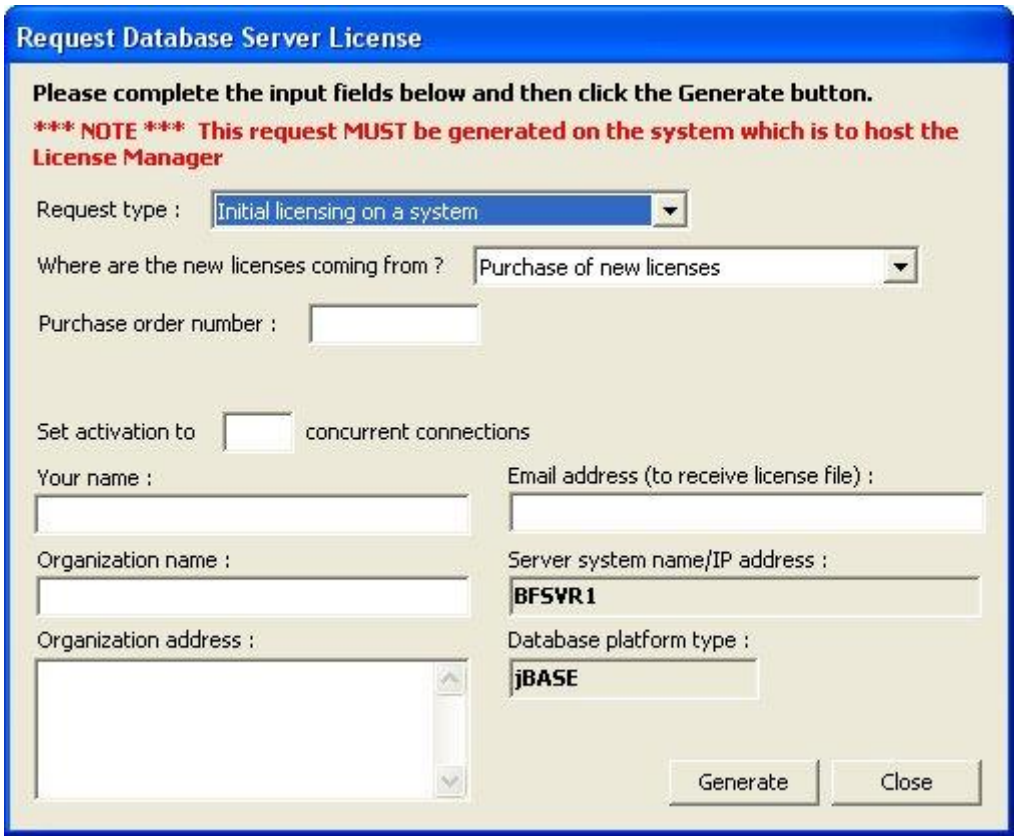


Diagram 8a : The Database Access License Request Window

On entering the appropriate details into the above input fields, you should click the 'Generate' button in order to be shown the text which you should email to your mv.NET supplier in order to receive the relevant database access license.

The same window will be shown if you use the License Request utility, except that you will need to manually enter the database platform type.

Installing a Database Access Licenses

Once a product license has been requested, you will receive an email containing a license file. On Vista, Windows7 and Server2008 systems, this file should be saved into the following location:

C:\ProgramData\BlueFinity\Licences

On all other systems, this file should be saved into the following location:

C:\Documents and Settings\All Users\Application Data\BlueFinity\Licences

Note, for Database Access Licenses, you need to perform this on the system which is hosting the License Manager service. For CID licenses, you need to do this on the system on which you have installed the CID product.

After saving the license file, please run the License Registration Utility – a shortcut to which will have been placed on your Start\BlueFinity\mv.NET menu by the mv.NET setup routine. This registration utility displays the following screen:

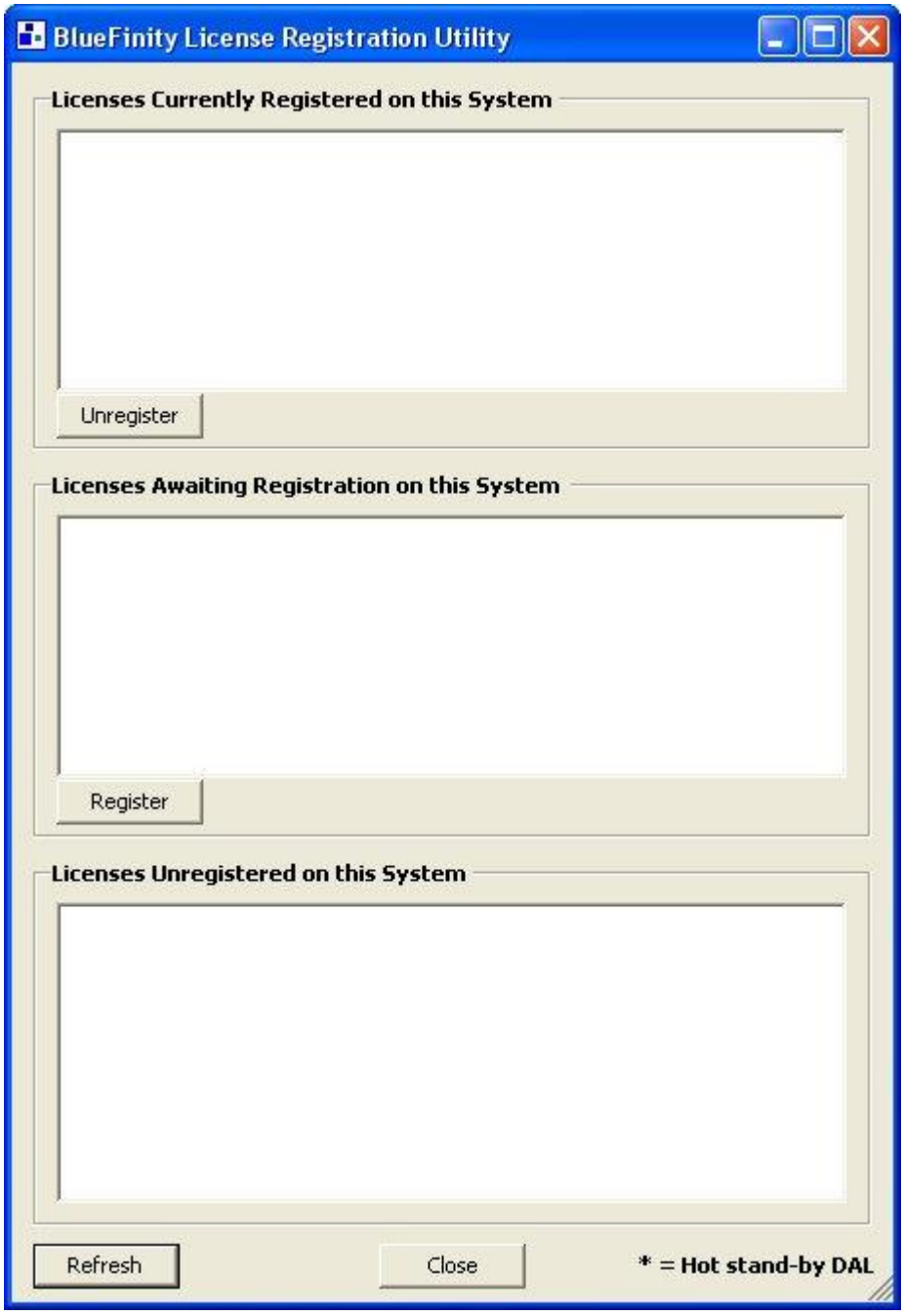


Diagram 8b : The License Registration Utility Window

If you already have a previously registered license for the same database (for example a previously activated evaluation license), you will need to highlight it within the top ListBox and click the Unregister button to remove it. Next, highlight the new license within the Licenses Awaiting Registration ListBox and click the register button. At this point your new license will be active.

Note, the License Manager does not need to be restarted for a new DAL license to be detected.

NOTE, it is essential that you do not amend the contents of any license files.

Viewing Installed Database Access Licenses

To view the database access licenses which are currently installed within the License Manager, you may use either the Data Manager or the Session Monitor applications. Both have a 'License Control' menu option, with a sub-option of 'View Database Licenses'. This option displays all the database access licenses installed within the License Manager.

License Manager Evaluation Mode

When the License Manager is first installed, it will, by default, allow up to 2 concurrent connections to any database installation for up to 30 days from the time of installation. Any access licenses applied during this evaluation period will, naturally, override this behavior. After the 30 days evaluation period has expired, access to databases will only be allowed if the appropriate database access license is installed.

Multiple License Managers

In situations where there are multiple session managers, you may introduce a 'secondary' License Manager to act as a fall-back capability should the system hosting the primary License Manger fail. The addresses of both License Managers should be entered into the Session Manager Settings node within the Data Manager explorer treeview. The secondary License Manager continually polls the primary to detect if it has gone off-line. If such a condition is detected, the secondary adopts the role of primary. The Session Managers also detects the loss of primary License Manager and automatically switches to

using the secondary. When the primary comes back on-line, the secondary and all Session Manager automatically revert to using it in preference to the secondary.

If a secondary License Manager is used, it must have all the equivalent license files installed as the primary. Please contact your mv.NET supplier for the supply of license files for secondary License Managers.

Class Library Overview

This Chapter provides an overview of the object classes contained within the Core Objects assembly (class library). These are the classes which the developer utilizes within their application code to access MultiValued systems from within their .NET application.

Introduction

Each class within the Core Objects class library contains a range of interface members – methods, properties and events. It is by using these members that the developer is able to carry out a wide range of MultiValued database related tasks within the .NET environment.

Many of the features provided by these classes will be familiar to the MultiValued developer; in fact, wherever possible, Core Objects' classes have been designed to mirror the traditional functionality provided by DataBASIC and other core MultiValued components.

However, becuase the client/server architecture differs in a number of fundamental ways from the legacy green screen application paradigm, mv.NET's classes provide features that will be new to the MultiValued developer. To this end, the Sample Application chapter following on from this class reference overview contains discussions covering several issues that the mv.NET developer should be aware of.

This class reference overview must assume a certain degree of knowledge on behalf of the reader – the main one being the understanding of how to program in .NET. If you need to obtain further information on this topic, please contact us at info@bluefinity.com and we will be happy to provide you with assistance on where you can obtain further help.

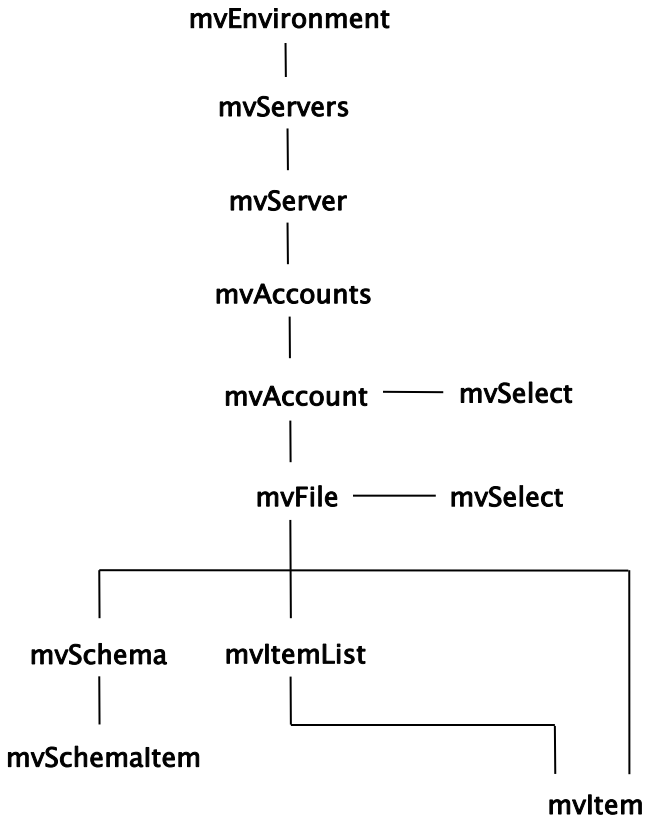
The examples provided in this guide are coded using VB.NET – however, you may use Core Objects within any .NET compliant language.

Core Objects - Class Summary

The Core Objects assembly contains the following main classes:

mvEnvironment	The top level class, primarily there to provide access to the Configuration Database.
mvServer	Represents a single MultiValued system that needs to be connected to. This class is rarely used, but exists primarily to provide a logically complete hierarchy.
mvAccount	Provides a connection to a specific MultiValued database account. This is the main class for accessing the contents of a MultiValued database.
mvFile	Provides a connection to a single database file.
mvItem	Holds a reference to a specific file item.
mvItemList	Hold a (selective) list of mvItem objects.
mvSchema	Allows maintenance of/access to schema definitions.
mvSchemaItem	Holds a reference to a specific schema item.
mvSelect	Allows advanced selection criteria to be supplied to selection-oriented methods of the mvAccount and mvFile objects.

The following diagram illustrates the hierarchical relationships between these classes:



The library also contains a number of additional ancillary classes:

mvDBRPC	Allows database server routines to trigger events within the .NET client environment.
mvDataTable	Inherits from the ADO.NET DataTable class. Provides ADO.NET style access to the data within an mvItemList instance.
mvQueryList	Represents the results of running a query defined using the Data Manager.
mvQueryRow	Represents a single row within an mvQueryList instance.
mvQueryColumns	Represents the columns within an mvQueryList instance.
mvQueryColumn	Represents a single column within an mvQueryColumns instance.
mvSessionControl	Allows programmatic control of the database

	session pooling.
mvSession	Represents an entry within the current database session pool.
mvSnapShots	Holds a list of mvItem instances representing a named snapshot of an item.
mvException	The standard exception class used internally by all Core Objects classes when an error is encountered.

Class mvEnvironment

An mvEnvironment instance may be viewed as the environment described by the local Configuration Database, i.e. an environment that is made up of MultiValued database servers, accounts, etc.

The creation and maintenance of an environment's definition, i.e. the creation and maintenance of Configuration Database entries, will typically be achieved using mv.NET's Data Manager application. This application, amongst many other things, provides any easy to use and intuitive interface for maintaining the contents of the Configuration Database. However, if you wish to maintain the Configuration Database programmatically you may do this via the mvEnvironment class.

To instantiate an mvEnvironment object you simply need to use the standard object instantiation method of your chosen .NET language; e.g. for VB.NET:

```
localEnv = New mvEnvironment
```

In the above code, the variable *localEnv* will be set to the local environment. As part of the object instantiation, the mvEnvironment object opens a connection to the local Configuration Database.

The mvEnvironment class constructor has an overload that allows you to specify programatically the location of the configuration database. This is the same string that would be present within the ConfigurationPath file. Zero-touch-deployment of an application is a classic example of where this approach to specifying the location of the configuration database would be used

As an example of this, say that the configuration database is on a server called Pluto located in a share called 'MainShare', within a folder called 'mv.NET'. The ConfigurationPath would thus need to be:

```
\\Pluto\MainShare\mv.NET
```

Note, there is no '\Configuration' at the end, this is automatically appended by mv.NET.

To use the mvEnvironment overload, the (VB.NET) code would be:

```
Dim myEnv As New mvEnvironment("\\Pluto\MainShare\mv.NET")
Dim myAccount As mvAccount = myEnv.Login("SOP") ' the login profile name
```

In C#

```
mvEnvironment myEnv = new mvEnvironment("\\Pluto\MainShare\mv.NET");
mvAccount myAccount = myEnv.Login("SOP");  // the login profile name
```

As an alternative to specifying the location of the configuration database via path, you can also specify it via the address of the Session Manager. In this case, all access to the configuration database goes via the Session Manager. If you wish to specify the location of the configuration database in this manner, it must be done using the form:

```
Address:Port    e.g.  Pluto:10013
```

Where *Address* is either the IP address or resolvable system name of the server which is hosting the Session Manager and *Port* is the port number on which the Session Manager is listening; i.e.:

```
Dim myEnv As New mvEnvironment("Pluto:10013")
```

Property Summary

The mvEnvironment class supports the following methods and properties (Type P = Property, M = Method):

Name	Type	Description
Connect	M	Allows an mvAccount object to be instantiated via a server and account profile name
ConnectedViaGateway	P	Returns a Boolean value indicating whether the environment instance has been created using a Gateway connection.
Configuration	P	Returns an mvConfiguration object allowing access to the Configuration

		Database files. See below.
GatewayURL	P	The URL of the Gateway being accessed.
LockingStyleDefault	P	Indicates whether Pessimistic or Optimistic locking should be the default style for all database sessions linked to this object.
Login	M	Allows an mvAccount object to be instantiated via a login profile name
PersistedGatewaySession		Indicates whether the Gateway being used supports persisted (stateful) sessions.
RemoteConfigDB	P	
SessionControl	P	Returns an mvSessionControl object allowing programmatic control of various aspects of session management.
SessionManagerAddress	P	The address of the Session Manager in the form *address:port*
SessionManagerHost	P	The system name or IP address hosting the Session Manager
SessionManagerPort	P	The port number on which the Session Manager is listening
SharedSessionActive	P	Returns a Boolean value indicating whether the specified server/account pairing is currently active as a shared connection.
Version	P	The version number of the referenced Core Objects library.

Class mvConfiguration

This class allows you to access and maintain the contents of the configuration database from within your own application code.

Member Summary

The mvConfiguration class supports the following interface members:

Name	Description
AccountNames	Returns a string array containing the names of all

	account profiles defined within the configuration database. There is an overload for this property that allows the account profiles for only a specific server profile to be returned.
Accounts	Returns an mvFile object allowing access to the account profiles defined within the configuration database.
Control	Returns an mvFile object allowing access to the system control information - primarily, the session manager settings.
GatewayNames	Returns a string array containing the names of all gateway profiles defined within the configuration database.
Gateways	Returns an mvFile object allowing access to the gateway profiles defined within the configuration database.
LoginNames	Returns a string array containing the names of all login profiles defined within the configuration database.
Logins	Returns an mvFile object allowing access to the login profiles defined within the configuration database.
ServerNames	Returns a string array containing the names of all server profiles defined within the configuration database.
Servers	Returns an mvFile object allowing access to the server profiles defined within the configuration database.
SetPassword	Allows the first password field of the specified account profile to be set.
SetPassword2	Allows the second password field of the specified account profile to be set.

The Account, Servers and Logins properties all allow access to the contents of the configuration database as if it were stored in a MultiValued database. The sections below describe the structure of items contained within these 3 files. Data may be accessed from these files using either attribute position or schema (dictionary) name.

Servers

The Servers property returns an mvFile instance allowing access to a file which contains items holding server profile definitions. The item ID of items within this file represents the name of each server profile. The structure of item data within this file is as follows (attributes marked with an asterisk are only relevant for IP-type connections):

Attribute#	Dictionary Name	Description
1	DatabaseType	The type of multivalued database represented by this server entry.
2	OSType	The underlying operating system hosting the mv implementation.
3	ConnectionType	The type of connection to be used to access the server.
4	ConnectionAddress	The connection specific address to be used to connect into the server.
5	ConnectionPort	The connection specific port number to be used to connect into the server.
*6	SendStrings	A multivalued list of character strings to transmit to the server during the connection process.
*7	WaitForStrings	A multivalued list of character strings that should be transmitted back from the server during the connection process.
*8	TimeoutPeriod	The maximum number of seconds to wait for any one WaitFor string to be received during the connection process.
*9	CommsDef	The low-level communication settings for the link. See below.
10	KeepAlive	A flag indicating how often (if at all) a keep alive heartbeat transmission should be sent to the server. '1' = true, '0' = false.
11	DisplayWindow	A flag indicating whether a connection window should be automatically displayed when a connection to this server is established. '1' = true, '0' = false.

Please refer to the previous server profile section for details on these fields.

The CommsDef attribute contains an encoded definition of the low-level communication characteristics for the IP connection to the database. The content of this attribute has the following format: CHnnn,mmm, where:

C = client to host characteristic
H = host to client characteristic

nnn = client to host chunk size
mmm = host to client chunk size

The client to host characteristic can have the following values:

A = Full 8-bit comms
B = 7-bit transmission of system delimiters
C = 7-bit transmission of control characters
D = 7-bit transmission of system delimiters and control characters
E = Full 7-bit comms

The host to client characteristic can have the following values:

A = Full 8-bit comms
B = 7-bit transmission of system delimiters

Please refer to the communication characteristics section for more details on this topic.

Accounts

The Accounts property returns an mvFile instance allowing access to a file which contains items holding account profile definitions. The item ID of items within this file takes the form: servername\accountname, where servername represents the name of a server profile and accountname represents the name of an account profile within the server profile. The structure of item data within this file is as follows:

Attribute#	Dictionary Name	Description
1	Account	The 'Account' login parameter.
2	SessionPoolMin	Minimum session pool size.
3	SessionPoolMax	Maximum session pool size.
4	SessionPoolCon	Maximum number of connections.
5	SessionPoolTerm	Session inactivity termination setting.
6		<reserved for future use>
7	PurgeSelAge	Select list clear down setting.
8		<reserved for future use>
9	PurgeImageAge	Read image clear down setting.
10	User	The 'User' login parameter.
11	Password	The 'Password' login parameter (read-only).

12	AutoConnect	The connect on session manager startup flag. '1' = true, '0' = false.
13	ShareMode	The session sharing mode. 0 = None, 1 = Single session, 2 = Multiple sessions.
14	QueueForSession	The Queue for free session flag. '1' = true, '0' = false.
15	QueuePoll	The queuing poll interval setting.
16	Prompt1	The 'Prompt1' login parameter.
17	Prompt2	The 'Prompt2' login parameter.
18	Password2	The 'Password#2' login parameter (read-only).
19	TmpFileSize	The temporary file size setting.
20	TmpFileDelPeriod	The temporary session file deletion period setting.
21	CmdTimeout	The default command timeout setting.
22	CacheSchema	The file schema caching flag. '1' = true, '0' = false.
23	FreeUpSession	The free-up allocated sessions setting.

Please refer to the previous account profile section for further details on these fields.

Logins

The Logins property returns an mvFile instance allowing access to a file which contains items holding login profile definitions. The item ID of items within this file represents the name of each login profile. The structure of item data within this file is as follows:

Attribute#	Dictionary Name	Description
1	ServerName	The name of the server profile to use.
2	AccountName	The name of the account profile to use.

Control

The Control property returns an mvFile instance allowing access to a file which contains various items of control data. The main item here is the session manager settings item, which has an ID of "SessionManager'. The structure of this item is as follows:

Attribute#	Description
1	<reserved>
2	The session manager address
3	The session manager listening port for v2.0 installations
4	Monitor message path
5	Message logging level
6	Housekeep run-time
7	Auto re-start flags
8	Auto re-start time
9	Housekeep on auto re-start flag
10	The session manager listening port for v2.1 installations
11	The session manager listening port for v3.1 installations
12	The session manager listening port for v3.2 installations
13	The session manager listening port for v4.1 installations
14	The session manager listening port for v3.5 installations
15	The session manager listening port for v4.0 installations
16	Maximum number of sessions per cluster
17	Maximum number of clusters
18	Maximum number of concurrent session launches
19	Session Manager recycle period (not used)
20	Session Cluster recycle period (not used)
21	Core Monitor recycle period (not used)
22	Independent housekeeping flag
23	Synchronized housekeeping flag
24	Session Manager 1 address
25	Session Manager 2 address
26	Session Manager 3 address
27	Session Manager 4 address
28	License Manager 1 address
29	License Manager 2 address
30	Logging delete period
31	Statistics delete period

32	Maximum concurrent Connection Monitor instances

These fields are the same ones as maintained by the 'Session Manager Settings' node within the Data Manager. Please refer to the Data Manager chapter for more information about these fields.

Gateways

The Gateways property returns an mvFile instance allowing access to a file which contains items holding gateway profile definitions. The item ID of items within this file represents the name of each gateway profile. The structure of item data within this file is as follows:

Attribute#	Description
1	The URL of the gateway.
2	The login profile name to be used at the remote gateway.
3	Persisted session flag. 0 = Non-persisted, 1 = persisted

These fields are the same ones as maintained by the Gateway nodes within the Data Manager. Please refer to the Gateway chapter for more information about these fields.

Class mvServer

This class allows access to the configuration details of a specific server profile and allows access to the accounts held on the server.

The Item method of the mvEnvironment.Servers property returns an mvServer object.

It is unlikely that you will need to use the mvServer class directly, except for maybe easy retrieval of server profile details. It is mainly there to provide a complete hierarchy with the class library.

Property Summary

The mvServer class supports the following properties:

Name	Description
Account	Returns an mvAccount object, the Login method of which may then be used be establish a database connection.
ConfigurationDefinition	Returns an mvItem object representing the corresponding server profile entry with the Configuration Database.
Environment	Returns the parent mvEnvironment object.

Class mvAccount

This class represents a specific account on a MultiValued system. It is by using this class' methods and properties that access to database information is achieved. An application may utilize several mvAccount objects concurrently.

A connection to an account can be established in one of 2 ways:

By supplying connection information as parameters to the mvAccount object constructor, e.g.

```
Dim myAccount As New mvAccount(LoginProfileName)
mvAccount myAccount = new mvAccount(LoginProfileName);
```

Or, by using either the Login or Connect methods of the mvEnvironment class, e.g.

```
Dim myAccount as mvAccount = myEnvironment.Login(LoginProfileName)
mvAccount myAccount = myEnvironment.Login(LoginProfileName);

Dim myAccount as mvAccount = myEnvironment.Connect(Server, Account)
mvAccount myAccount = myEnvironment.Connect(Server, Account);
```

Note, using the mvEnvironment class to obtain an mvAccount instance is slightly more efficient in situations where mvAccount instances are being instantiated on a frequent basis. This is because the mvAccount constructor automatically creates an mvEnvironment instance internally during its construction.

The mvAccount constructor and the mvEnvironment Login and Connect methods all contain a number of overloads which optionally allow the Session Manager address, username, password, applicationGUID and client ID to be passed, these are explained below:

Session Manager address: This is the address of the Session Manager. This must be passed if there is not a local configuration database or ConfigurationPath file accessible on the local system. This must be passed in the form *address*:*port*

UserName: The user name to be used during the connection sequence. If this is not required or is defined as part of the server/account profile it should be passed as an empty string ("") value.

Password: The password to be used during the connection sequence. If this is not required or is defined as part of the server/account profile it should be passed as an empty string ("") value.

ApplicationGUID: An optional unique identification of this application instance. This is only required in situations where item lists need to be retained within stateless environments, e.g. web applications.

ClientID: The user-definable identification of this application instance. If left blank, this defaults to: `system name:process name:processID`

Method/Property Summary

The mvAccount class supports the following methods and properties (Type P = Property, M = Method):

Name	Type	Description
AppSessionGUID	P	The unique identifier for this login session.
CallProg	M	Calls a cataloged MultiValued DataBASIC subroutine.
CallProgViaExecute	p	Boolean which when True indicates that the CallProg method is to be performed within a new execute level on the database server. This can help avoid common area conflicts.
CommandTimeoutPeriod	P	The default maximum wait period for a response back from the MultiValued server upon issuing any form of request/action.

		Defaults to 30 seconds.
Connected	P	Indicates whether this mvAccount instance holds a connection to a database session,
Environment	P	The mvEnvironment object associated with this instance.
Execute	M	Executes any command-line statement.
FileClear	M	Clears all items in a file
FileCount	M	Counts the number of items in a file (optionally with selection criteria).
FileCreate	M	Creates a new file.
FileDelete	M	Deletes a file.
FileList	M	Returns the list of visible file names within the account.
FileOpen	M	Opens a specific file and, if successful, returns an mvFile object connected to that file.
FileOpenBool	M	Same as FileOpen method but, additionally, returns a boolean value indicating success or failure.
FileVisibility	P	Allows programmatic control over whether a file should appear in the FileList property.
ListDelete	M	Deletes a previously stored save-list.
ListGet	M	Retrieves a previously stored save-list in to an mvItemList object.
ListRestore	M	Re-establishes a previously saved mvItemList (for stateless applications).
LockingStyleDefault	P	The default locking scheme to be used by all methods relating to the creation and release of item locks.
Login	M	Establishes a login session to the account.
Logout	M	Logs out (disconnects) from the account.
Name	P	The name of the account (i.e. the name of the account profile used to connect into the account).
Port	P	The MultiValued server port number associated with this session.

ProgSelect	M	Allows a cataloged MultiValued DataBASIC subroutine to be used to supply the required list of item IDs. This subroutine must support 3 arguments: 1. INPUTARG – the string passed into the ProgSelect method call. Structure and use of this to be decided by the developer but **MUST NOT** contains AM or VM characters 2. ITEMIDS – a returned AM delimited list of required item IDs 3. ERRMSG – to be set to a non-blank value if an error is encountered, i.e. a description of the error.
QueryRun	M	Executes a predefined query, returning an mvQueryList object.
ReleaseAllLocks	M	Releases all item locks for the host MultiValued system.
RetrievalSizeDefault	P	The default initial retrieval size for selection actions. Defaults to 200.
RetrieveIDsOnlyDefault	P	Sets/returns whether only items IDs should (by default) be initially retrieved during selection actions. Defaults to False.
SchemaCaseSensitivity	P	Allows control over whether reference to field data via schema name is case sensitive.
Select	M	Executes any command-line statement that exits with an active a select list.
Server	P	The parent mvServer object.
SessionSharing	P	Indicates whether the associated account profile is defined to use session sharing.
TransAbort	M	Rolls-back all transactions performed within the current transaction boundary.
TransCommit	M	Commits all transactions performed within the current transaction boundary.
TransQuery	P	Indicates whether a transaction boundary is active.

TransStart	M	Initiates the opening of a transaction boundary.

Class mvFile

This class represents a single database file (dictionary or data portion). It is by the use of this class' methods and properties that access to all dictionary and data information within an individual file is achieved.

An mvFile instance is obtained by using the FileOpen method of the mvAccount class.

Method/Property/Event Summary

The mvFile class supports the following methods and properties (Type P = Property, M = Method, E=Event):

Name	Type	Description
Account	P	The parent mvAccount object.
BulkUpdateAbort	M	Indicates that bulk updating mode is to be aborted. Any outstanding bulk update writes are lost.
BulkUpdateErrors	E	Raised if bulk updating writes generate a database error.
BulkUpdateFinish	M	Indicates that bulk updating mode is to be exited. Any outstanding bulk update writes are written to the database.
BulkUpdateInProgress	P	Indicates whether bulk updating mode is active.
BulkUpdateSize	P	The number of items to be written to the database on each round-trip. Defaults to a value of 200.
BulkUpdateStart	M	Indicates that bulk updating mode is to be started. See 'Additional Notes' section below for more details.
Clear	M	Deletes all items from the file.
Count	M	Counts the number of items in the file (optionally with selection criteria).
Delete	M	Deletes a specific item from the file.
IndexSelect	M	Returns an mvItemList object representing all or a subset of items (and associated data) within the

		file as ordered by a secondary index.
NewItem	M	Returns a new mvItem object associated with the file (note, no item is created within the database until this item is explicitly written).
Name	P	The name of the file; e.g. CUSTOMERS or DICT PRODUCTS
ProgSelect	M	Allows a cataloged MultiValued DataBASIC subroutine to be used to select items into an mvItemList.
QSelect	M	Uses the contents of an item (or part of) as the source of item IDs. It returns an mvItemList object based on the retrieved list of item IDs.
Read	M	Reads (without locking) a specific item from the file, returning a reference to an mvItem object (holding the item data).
ReadBool	M	Same as Read method but returns a boolean value indicating success or failure.
ReadU	M	Same as Read method, but also places a lock on the item.
ReadUBool	M	Same as ReadBool method, but also places a lock on the item.
ReadV	M	Reads (without locking) a specific attribute number from a specific file item.
ReadVBool	M	Same as ReadV method but returns a boolean value indicating success or failure.
ReadVU	M	Same as ReadV method, but also places a lock on the item as a whole.
ReadVUBool	M	Same as ReadVU method but returns a boolean value indicating success or failure.
Release	M	Releases one or all item locks for the file.
Schema	P	Returns an mvSchema object representing the associated schema definitions of the file.
Select	M	Returns an mvItemList object representing all or a subset of items within the file.
Write	M	Writes an mvItem object to the file, releasing any locks held on the specified item ID.
WriteU	M	Same as Write, but without the lock release action.

WriteV	M	Writes a value to a specific attribute number within a specific file item, releasing any locks held on the specified item ID.
WriteVU	M	Same as WriteV, but without the lock release action.

Additional Notes

The following sections provide additional information relating to the use of the above mvFile class interface members

Bulk Updating

Bulk Updating allows you to optimize the writing of database items by reducing the number of round-trips to the database server. Instead of each write action resulting in a round-trip, write details are buffered internally by mv.NET at the client and are only sent to the database once a (user-definable) number have occurred.

Below is a VB.NET code snippet which shows the use of bulk updating:

```
Dim WithEvents myFile As mvFile

Sub Test()

    myFile = myAccount.FileOpen("ORGANIZATION")
    myFile.BulkUpdateSize = 100
    myFile.BulkUpdateStart()
    ' ... perform lots of mvItem.Write actions here
    myFile.BulkUpdateFinish()

End Sub

Private Sub myFile_BulkUpdateErrors(ByVal ErrorItems As
System.Collections.ArrayList, ByVal ErrorMsgs As
System.Collections.ArrayList) Handles myFile.BulkUpdateErrors
    ' ... report/log errors here
End Sub
```

The above code illustrates the use of the bulk update start and finish methods before and after the required item writes. If not set explicitly, the BulkUpdateSize property defaults to a value of 200. You can use this property to fine tune the bulk updating process - a higher size results in fewer database server round-trips but consumes more client memory and results in larger message sizes being sent to the database.

The BulkUpdateErrors event is fired if any database write errors occur. The ErrorItems argument holds an ArrayList of mvItem objects representing the items which failed to be

written to the database. The ErrorMsgs argument holds a respective list of error messages for each corresponding entry in ErrorItems list.

Exploded Selecting

Since one of the primary features of a Multivalued database is to allow the nesting/repeating of data within a single record, an important requirement is to allow selections to be made against this nested data. Such selecting must support the concept of partial selection of nested data based on selection criteria; that is, to support the concept of the selection results, in effect, only returning partial data content of the overall selected items. This feature is known as 'exploded selecting' since most Multivalued databases explode (or flatten) the contents of items before performing the selection process. The resulting data is presented in flattened form with none qualifying multivalues omitted.

mv.NET supports the concept of exploded sorting via the SelectByExp method of the mvAccount and mvFile classes. The basic concept is that you specify which fields are to be exploded and the selection criteria to be applied during the selection process. It is assumed that all fields which are exploded are associated in that each of their multivalue lists are ordered and related in unison.

The result of the SelectByExp method is an mvItemList. This list presents the illusion of each row of exploded data being a separate item. However, internally, mv.NET tracks the fact that each item in the list is, in fact, an item fragment and if items are amended and written back to the database, mv.NET will only update the associated portion of the item on file. This, thus, gives the developer the best of both worlds - being able to access a filtered view of item data whilst still being able to update data content.

The SelectByExp method has several overloads, each of which is explained below.

```
Public Function SelectByExp(ByVal PrimaryExplodeField As String, ByVal
SecondaryExplodeFields As String) As mvItemList
```

This overload will select all items in the file, returning an exploded view based on the content of the PrimaryExplodeField argument.

```
Public Function SelectByExp(ByVal PrimaryExplodeField As String, ByVal
SecondaryExplodeFields As String, ByVal SortType As ExpSortType, ByVal
ExplodedSelection As String, ByVal NonExplodedSelection As String, ByVal
DictionaryList As String, ByVal Attributes As String) As mvItemList
```

This overload will select a sub-set of items. The ExplodedSelection argument needs to be set to the select clause which is to be applied to the PrimaryExplodeField. The NonExplodedSelection will be applied to the rest of the item content. The DictionayList and Attributes arguments allow partial data retrieval to be performed.

```
Public Function SelectByExp(ByVal PrimaryExplodeField As String, ByVal
SecondaryExplodeFields As String, ByVal SortType As ExpSortType, ByVal
ExplodedSelection As String, ByVal NonExplodedSelection As String, ByVal
DictionaryList As String, ByVal Attributes As String, ByVal SelectControl As
mvSelect) As mvItemList
```

This overload works exactly as the previous one except that the Preselection and UsingDict properties of an mvSelect instance can be utilized during the selection process.

Class mvItem

This class represents an multidimensional MultiValued dynamic array – typically representing an item within a file. As such, it interprets any embedded ASCII 254 (attribute mark), 253 (value mark) and 252 (subvalue mark) codes as special data element delimiters.

An mvItem instance will typically be obtained using the Read method of the mvFile class, although it also has a publicly accessible constructor that is overloaded to receive an initial string value.

Method/Property Summary

The mvItem class supports the following methods and properties (Type P = Property, M = Method, E = Event):

Name	Type	Description
AfterModified	E	Fires after the contents of an item have been modified.
AssignDefaults	M	Assigns default values to attributes within the item as defined by extended dictionary details.
BeforeModified	E	Fires just before the contents of an item are about to be modified.
Clone	M	Returns a new instance copy (clone) of the mvItem object.
Contents	P	The entire (raw) physical data content of the item.
Count	M	Counts the number of occurrences of a string within all or

		part of the item.
Data	P	Provides access to all sections of data within an item based on either physical storage position or dictionary name. This property is the default indexer for the class.
DataAsArray	M	Allows (read-only) access to data as an array. Intended for use within For-each loops.
DataAsString	M	Allows (read-only) access to data, returning values as strings, formatting in accordance with dictionary definitions.
DataRaw	M	Allows (read-only) access to raw (unformatted) data based on dictionary name.
DCount	M	Counts the number of occurrences of delimited cells within all or part of the item.
Delete	M	Deletes a specific element from the item.
DictionaryList	P	The list of dictionary field names whose values were retrieved along with the raw data of the item.
Dispose	M	Allows the server-resident read image data and locks for this item to be explicitly deleted/cleared.
Field	M	Returns one or more attribute/value/subvalue occurrences from with the item's data content.
File	P	The parent mvFile object (set automatically if the item was created via an mvFile IndexSelect, Read or Select method call.
GroupInsert	M	Allows a value to be inserted into a group of associated values.
GroupDelete	M	Allows a value to be deleted from a group of associated values.
IConv	M	Returns a value by applying a MultiValued input conversion code to a data value within the item.
ID	P	Returns the ID of the item.
Index	M	Returns the position of a character or characters within all or part of the item.
IndexEntry	P	Returns the index value associated with the item for items retrieved via an mvFile's IndexSelect method.
Insert	M	Inserts elements into the item's contents.
InvalidatedCalculations	P	The list of calculated values that are currently invalid due to dependent data change.

ItemList	P	The parent mvItemList (for items retrieved via a selection method).
Locate	M	Finds the position of an element within a specified dimension of the item.
Lock	M	Places a lock on the item.
LockStyle	P	The style of locking used when the item was read.
Modified	P	Indicates whether any of the item's data content has been amended since it was originally read or last written (whichever is the most recent).
OConv	M	Returns a value by applying a MultiValued output conversion code to a data value within the item.
QSelect	M	Uses the contents of all or part of the item as the source of item IDs. It returns an mvItemList object representing the supplied list of item IDs.
Recalc	M	Regenerates any dictionary-derived data. A list of dictionary names can be passed with this method to restrict the action to a subset of calculated fields.
Refresh	M	Re-reads the data of the item. If dictionary-derived data is being held, this is also regenerated.
Release	M	Release the lock on the item.
Replace	M	Replaces a specified data element within the item.
Sort	M	Allows the contents of an item to be sorted.
Text	P	Same as the Data property, but with data cell delimiters replaced with specified character(s).
Write	M	Writes the item to a file, releasing any locks held on the specified item ID. This is only available if the File property contains a reference.
WriteU	M	Same as Write, but without the lock release action.
Unlock	M	Same as Release method.

Class mvSelect

This class is provided in order to allow advanced selection details to be conveniently supplied to the following methods:

```
mvAccount.Select
mvAccount.ProgSelect
mvAccount.ListGet
mvFile.IndexSelect
mvFile.QSelect
mvFile.Select
```

The provision of an mvSelect object to any of the above method calls is optional, but if present, allows a simple, powerful and flexible way of specifying advanced selection criteria. Most of the properties supported by the mvSelect object are optional, only the ones of relevance to the selection context need be set.

Usage Syntax

The following code illustrates the use of an mvSelect object in a Select method call of an mvFile object:

```
Dim Customers As mvFile
Dim UKCustomers as mvItemList
Dim criteria as New mvSelect

Customers = SOPAccount.FileOpen("CUSTOMERS")

criteria.SelectionClause = "WITH COUNTRYCODE = ""44"""
criteria.DictionaryList = "PRIMARYCONTACT PRIMARYTELNO"
criteria.RetrievalStyle = RetrievalStyle.PartialDataInitiallyRestOnDemand
criteria.RetrievalSizeInitial = 100
criteria.RetrievalSizeOnDemand = 50

UKCustomers = Customers.Select(criteria)
```

The above code opens the CUSTOMERS file and then selects all items from this file with a COUNTRYCODE attribute value of '44'. After the items have been selected, the item data (along with the values of the PRIMARYCONTACT and PRIMARYTELNO dictionary fields) of the first 100 items are returned – the rest of the selection data being held on the server. As data is consumed by the client, additional selection data is retrieved as required from the server in batches of 50 items at a time.

Property Observance

Not all properties of the mvSelect class are used/observed by all selection methods. Below is a list of the properties which will be used by each of the possible selection methods.

mvAccount.Select

The mvAccount.Select method observes all of the mvSelect properties except for SelectClause and SortClause. This is because it is assumed that the SelectCommand argument of this method contains a fully formed selection command containing the required selection and sort criteria.

mvAccount.ProgSelect

The mvAccount.ProgSelect method observes all the mvSelect properties. If the PreSelection property is supplied, it will be applied to the item IDs returned by the user supplied subroutine.

mvAccount.ListGet

The mvAccount.ListGet method observes the following mvSelect properties:

Attributes
DictionaryList
FileName
IDsOnly
RetrievalSizeInitial
RetrievalSizeOnDemand
RetrievalStyle

mvFile.IndexSelect

The mvAccount.Select method observes all the mvSelect properties except for the following:

FileName
SelectClause
SortClause
PreSelection
SaveListName
SaveListOnly

mvFile.Select

The mvFile.Select method observes all the mvSelect properties.

mvFile.QSelect

The mvAccount.QSelect method observes all the mvSelect properties. If the PreSelection property is supplied, it will be applied to the supplied list of item IDs.

Property Summary

The mvSelect class supports the following properties

Property Name	Description
AllowNoItems	Allows control over whether an exception is raised if no items are selected.
DisableCaching	Forces item data to be continually read on demand, with no internally caching of items read. If set to True, internal memory resource usage is kept to a minimum (at the cost of retrieval performance).
DictionaryList	The (space delimited) list of dictionary field names whose values are to be retrieved along with the raw data of the item.
FileName	The name of the file to process.
IDsOnly	Boolean value indicates whether only item IDs are to be returned as a result of the selection.
PreSelection	A command level statement that pre-selects the list of items IDs to be processed. This can be any command that exits leaving an active select list.
RetrievalInterval	The frequency of background item list population.
RetrievalSizeInitial	The number of items to be returned to the client initially on completion of the selection process.
RetrievalSizeOnDemand	The number of items to be returned to the client on each subsequent round-trip to the server to retrieve further selected items.
RetrievalStyle	The style of data retrieval to be used.
RetrievalThreads	The number of additional threads that are to be used to retrieve data in the background. This is only relevant for a RetrievalStyle setting of PartialDataInitiallyRestInBackground
SaveListName	The name of the save-list to generate as a result of the selection
SaveListOnly	Boolean value indicating whether only a save-list is to be generated - i.e. no data is to be returned.
SelectionClause	The selection criteria to apply during the selection process.

SortClause	The sort criteria to apply during the selection process.
UsingDict	The name of the alternative dictionary to use in the selection.

Class mvItemList

This class represents a sequence of items selected from a file. It allows this list to be traversed sequentially or by absolute position.

Method/Property Summary

The mvItemList class supports the following methods and properties (Type P = Property, M = Method) :

Name	Type	Description
Add	M	Adds an mvItem object at a specified position within the list.
Calculate	M	Allows a field within the item list content to be totaled, averaged as well as minimum and maximum value calculated.
ClearCache	M	Clears all cached items within the list.
Clone	M	Creates a new instance copy (clone) of an mvItemList object.
Count	P	Returns the total number of selected items. If the mvItemList object has been created via an mvFile's IndexSelect method, the number of items that have been read from the index so far is returned.
CurrentItem	P	The item indicated by the current cursor position within the list.
CursorPos	P	The current cursor position within the list.
DataTable	M	Returns an mvDataTable object. See mvDataTable section below
DictionaryList	P	The list of dictionary field names whose values were retrieved along with the raw data of items.
EOL	P	Indicates whether the end of the list has been reached.
File	P	The parent mvFile object (set automatically if the item list was created via a selecting method call.

ID	P	Returns the item ID of an mvItem object at a given position within the list.
IndexDerived	P	Indicates whether the item list has derived from an index (True) or if it has been derived from a selection (False).
IndexRetrieved	P	Indicates whether a specific item within the list has been retrieved into the list cache.
Item	P	Returns an mvItem object at an absolute position within the list.
MoveNext	M	Moves the list's cursor to the next item in the list.
MovePrev	M	Moves the list's cursor to the previous item in the list.
Persist	P	Indicates whether the item list should be preserved on the server when the mvItemList object is destroyed.
ReadNext	M	Retrieves the next mvItem entry in the list.
ReadNextBool	M	Retrieves the next mvItem entry in the list returning a boolean value indicating success or failure.
ReadNextID	M	Retrieves the next item ID in the list.
ReadPrev	M	Retrieves the next mvItem entry in the list.
ReadPrevBool	M	Retrieves the next mvItem entry in the list returning a boolean value indicating success or failure.
ReadPrevID	M	Retrieves the next item ID in the list.
Remove	M	Removes (and optionally deletes from the database) an mvItem object in the list at a specified position.
Reset	M	Repositions the list's cursor to the first entry in the list.
RetrievedSoFar	P	The number of items physically retrieved into the item list so far.
SelectControl	P	Returns the mvSelect object that was used to control the selection process used to create the item list.
State	P	Sets/returns the current state of the item list. This can be used to persist/reinstate item selections in stateless environments.
WriteModifiedItems	P	Writes all items within the list that have a Modified property value of True to the database.

Class mvSchema

This class represents a file's dictionary schema – it is exposed by the Dictionary property of the mvFile class. It is different to an mvFile object which has been opened on the dictionary portion of a file in that the mvSchema class presents the dictionary as a collection of (highly structured) mvSchemaItem objects – see next section for more details on the mvSchemaItem class.

Method/Property Summary

The mvSchema class supports the following methods and properties (Type P = Property, M = Method) :

Name	Type	Description
Add	M	Adds a new mvSchemaItem object to the collection (and optionally writes its content to the dictionary portion of the associated database file).
BooleanTest	M	Allows a test to be performed to check whether a specified value is regarded as a True or False value for a specified dictionary name.
ContainsKey	P	Indicates whether a specified dictionary name exists in the collection.
Item	P	Returns an mvSchemaItem occurrence from the collection based on either position or name.
ItemIDField	P	Returns the mvSchemaItem which represents the item ID of the associated file.
Load	M	Forces the collection of mvSchemaItem objects to be reloaded from the database.
MembersOf	M	Allows the names of all dictionary items belonging to a specified mv or sv group to be obtained.
MVGroupOf	M	Allows the MultiValued group of a specified dictionary name to be obtained.
NamesFromAttrPos	M	Allows the list of dictionary name representing a specified attribute to be obtained.
NewItem	M	Returns a new mvSchemaItem (which is automatically mapped to this mvSchema).
Remove	M	Removes a specified mvSchemaItem from the collection (and optionally from the associated database file).

Class mvSchemaItem

This object represents a dictionary definition item within the dictionary portion of a file. It may be used to access both the native and extended dictionary definitions of a field.

Extended Dictionary Definitions

Each native dictionary definition within the dictionary portion of a file allows users and developers to refer to the attributes of data within an item by name as well as physical storage position. These native definitions are obviously used by many of the tools and commands available on a MultiValued platform.

mv.NET, however, allows you to augment the relatively limited set of definition data within native definitions with what are known as *extended dictionary definitions*. An extended dictionary definition is stored within the same dictionary portion of a file under the same ID as the native definition that it extends but with curly braces at the start and end of the ID. For example, native dictionary item NAME would have its extended definition stored as {NAME}.

Extended definitions are only used by mv.NET and are used in conjunction with native definitions, NOT instead of – native definitions are still very important to mv.NET. The extra definition data within extended definitions allows various aspects of mv.NET to better understand the structure and type of data within an item, the relationships between attributes of data within the same item and also between items of data across different files.

It is not necessary to create extended definitions to use mv.NET, but as you start to use the more powerful aspects of the product, you will find it worthwhile to invest a little time and effort in creating extended definitions.

The easiest way to create extended dictionary definitions is by using the Schema Editor within the Data Manager, however, you can also create and maintain both native and extended definitions using the mvSchema and mvSchemaItem classes within Core Objects.

For further information on extended dictionary definitions please refer to the chapter dedicated to this topic.

Referring to Attributes by Name

The mvItem class within Core Objects provides a number of properties allowing you to supply either an attribute position or a dictionary name in order to refer to a particular field of data; an example of this being the Data property.

If you supply a dictionary name to one of these properties, Core Objects will follow the following list of steps to ascertain which physical piece of data is being referred to:

1. Check if the specified dictionary name is in the list of item values that have been retrieved by name in the original fetch of item data. If so, return the retrieved data field.
2. Load the dictionary portion of the file if it has not already been loaded. Note, the FileOpen method of the mvAccount class allows you to explicitly request the load of a file's dictionary at the same time as opening it, otherwise it will be done on demand. The loading of a file's dictionary will result in both native and extended definitions being passed to the client. It is the 'loading' of a file's dictionary that results in the mvSchema property of an mvFile object being populated.
3. If the specified dictionary name is not found within the loaded dictionary, raise an exception.
4. If the specified dictionary name refers to a non-calculated field of data, use the definition to retrieve the relevant value.
5. Perform a round-trip to the server to retrieve the specified value based on dictionary name (this will force the native dictionary definition to be used on the server to retrieve the appropriate value).

Method/Property Summary

The mvSchemaItem class supports the following methods and properties (Type P = Property, M = Method). Note, all properties that start with 'Input' are only used by the data binding aspects of the Binding Objects components and are not used when values are assigned to an mvItem instance programmatically.

The mvSchemaItem will interpret the contents of a native definition as best it can, and, in the absence of an extended item, will provide a limited set of properties describing the field. In the summary table below, the property names flagged with an asterisk are the properties that will be supported if only a native definition item is found. All the other properties are only supported if an extended definition item is present.

Name	Type	Description
AttrPos *	P	The physical attribute position represented by this dictionary item. Returns 0 for calculated items.
BaseDependencies	P	The list of non-calculated dictionary items that this field is dependent upon. Returns an attribute delimited string of names. Only relevant for calculated fields.

BooleanFalse	P	The string value that represents a logical True value for this field.
BooleanTrue	P	The string value that represents a logical false value for this field.
ConversionCode *	P	The conversion code that will be applied to the raw item value to generate the displayable value. This code is also used in the opposite direction, i.e. the conversion of a displayable value to a raw (internal) item value.
CorrelativeCode *	P	The code that will be applied to the raw item value to generate the value to be used in any file processing commands.
DataType *	P	The data type of the field. See following Common Enumerators section.
DataTypeCheck	M	Returns a Boolean value indicating whether the supplied value represents a valid data type for this field.
Dependencies	P	The complete list of dictionary items that this field is dependent upon. Returns an attribute delimited string of names. Only relevant for calculated fields.
Extend	M	Forces the initial set of extended properties for this item to be generated based on the native definition.
ExtendedDefinition	P	The raw extended definition item.
GUID	P	A unique identifier for this dictionary item independent of name. This is assigned on dictionary load.
InOptionsCheck	M	Returns a Boolean value indicating whether the supplied value represents a value that is present within the InputOptions property.
InputCaseAdjust	M	Adjusts a supplied value in accordance with the setting of the InputCasing property setting.
InputCasing	P	The upper/lower casing specification for the field. See following Common Enumerators section.
InputDefault	P	The default value for the field.
InputInOptions	P	Indicates whether any supplied values for this field have to be present within the InputOptions

		property.
InputMandatory	P	Indicates whether this field may be left blank.
InputMax	P	Indicates the maximum allowable value for this field. This property returns a string representation of this value.
InputMaxValue	P	Indicates the maximum allowable value for this field. This property returns an object cast according to the DataType of the field.
InputMin	P	Indicates the minimum allowable value for this field. This property returns a string representation of this value.
InputMinValue	P	Indicates the minimum allowable value for this field. This property returns an object cast according to the DataType of the field.
InputOptions	P	The list of valid values for this field. This returns a MultiValued list of valid values.
InputPrompt	P	The default input prompt for the field.
IsCalculated	P	Indicates whether the field represents a calculated (derived) value.
IsExtended	P	Indicates whether an extended definition exists for this field.
Justification *	P	The default display justification for this field. See following Common Enumerators section.
LinkedFile	P	The file to which this field holds item Ids.
LinkedFileCheck	M	Returns a Boolean value indicating whether the supplied value represents a valid item ID within the file indicated by the LinkedFile property.
LinkedFileDescField	P	The default descriptive attribute within associated item within the linked file.
LinkedFileIDField	P	The attributes representing the item ID within the linked file.
LinkedFileProperty	P	The name to be used for a property relating to the linked file within Solution Objects.
Modified	P	Indicates whether (non-saved) modifications to the loaded definition exist.
MaxValueCheck	M	Returns a Boolean value indicating whether the supplied value represents a value which is equal to or less than the InputMaxValue property.

MinValueCheck	M	Returns a Boolean value indicating whether the supplied value represents a value which is equal to or less than the InputMinValue property.
MVGroup	P	The logical (associative) multivalue group to which this field belongs.
MVType	P	Indicates whether this field is MultiValued. See following Common Enumerators section.
Name *	P	The name of the dictionary item.
NativeDefinition *	P	The raw item string of the native definition.
NativeType *	P	Indicates the type of native definition, e.g. 'A' type definition, or 'V' type definition.
RecalcDef	P	The local recalculation definition.
RecalcLocal	P	Indicates that the RecalcDef property should be used to recalculate the value of the field if any of its base fields (the fields upon which it depends) change.
Schema	P	The parent mvSchema object for this mvSchemaItem instance.
SVGroup	P	The logical (associative) subvalue group to which this field belongs.
Title *	P	The display (report) title of this field.
ToString	P	Returns the same value as the Name property.
U2FORMAT *	P	For UniData and Universe style definitions (D, I and V types), this property returns attributes 5 of the definition.
Width *	P	The display (report) width of the field.
Write	M	Writes the dictionary definition back to the database.

Class mvDBRPC

This class can be used to implement database server triggered calls to .NET resident components. Examples of this might be:

- Generating documents via a Windows based application when certain events occur, or options are chosen within the database server environment or green-screen application.

- Moving data from a MultiValued database to SQL when certain events occur or a certain period of time elapses.

There are 2-parts to implementing database remote procedure calls. The first is to instantiate one or more mvDBRPC instances within the .NET environment, i.e. within your .NET application. Doing this will initiate polling against a specified file or files (termed the "trigger" file) on the database server. Database server routines are then able to write items to the trigger file and these will be automatically picked up by the polling mvDBRPC instances which will then raise an event to indicate the appearance of these items.

The code snippet below illustrates the instantiation of an mvDBRPC object and the handling of the event which is raised when an item appears in the 'DBRPC' database file:

```
Dim RPCTracker As New mvDBRPC("SOP", "DBRPC", 1000, 3)
AddHandler RPCTracker.RPCRequested, AddressOf RPC

Private Sub RPC(ByVal Sender As mvDBRPC, ByVal RPCItems As mvItemList)

    For Each rpc As mvItem In RPCItems
        ' Perform some really useful action here
        '
        Console.WriteLine("RPC:" & rpc.Contents)
        Sender.RPCCompleted(rpc.ID)
    Next

End Sub
```

In the above code, the arguments to the mvDBRPC constructor indicate that the DBRPC file in the SOP account is to be scanned every 1000 milliseconds for the appearance of new items. When a new item appears, its contents will be passed via the RPCItems argument of the RPCRequested event. The last argument in the constructor ("3") indicates that a maximum of 3 RPCs are to be collected with each poll of the server. A value of -1 indicates all new RPCs will be collected. It is by using this last argument that multiple processes or threads may work in unison to improve RPC throughput performance.

The call to the RPCCompleted method of the mvDBRPC object will result in the item being deleted from the DBRPC file. The server routines may thus, if required, detect successful execution of the rpc.

If the rpc fails, the RPCFailed method may be called, which will result in an attribute containing the string "RPC Failed : " and a description of the error being concatenated to the front of the rpc item within the trigger file.

The mvDBRPC class also supports the concept of "throttling back" the frequency of polling against the trigger file. At the point in time where a specified number of contiguous polls of

the trigger file have returned no items to process, the frequency of polling can be adjusted to occur at longer intervals. When a trigger item is subsequently found, the frequency of polling is reset back to the original (more frequent interval) and whole process starts again.

Method/Property Summary

The mvDBRPC class supports the following interface members (Type P = Property, M = Method, E = Event) :

Name	Type	Description
Account	P	The mvAccount instance being used or to be used by the mvDBRPC instance for its database polling activity. This property will return null if an mvAccount instance was not passed as part of its constructor.
RPCFileName	P	The name of the database file where RPCs will be created.
LoginName	P	The login profile name used to connect to the required database account.
PollInterval	P	The number of milliseconds to wait between polls of the database RPC file.
PollIntervalAfterThrottleback	P	The (longer) polling interval (in milliseconds) to be used after throttling back has occurred – see ThrottleBackThreshold property. The value of this property defaults to PollInterval * 5
MaxRPCsToProcess	P	The maximum number of RPCs to extract from the server.
RPCCompleted	M	Allows successful completion of the RPC to be communicated back to the database server.
RPCFailed	M	Allows unsuccessful completion of the RPC to be communicated back to the database server.
RPCRequested	E	Indicates that one or more new RPCs have been detected within the RPC file.
RPCError	E	Indicates that the mvDBRPC object has encountered an internal error.
ThrottleBack	M	Allows throttling back to be activated programmatically.
ThrottleReset	M	Allows throttling back to be cancelled

		programmatically.
ThrottleBackThreshold	P	Indicates when the longer poll interval (held in PollIntervalAfterThrottleback) should kick in. It indicates the number of contiguous (zero-item returning) scans of the trigger file after which the poll interval will be set to PollIntervalAfterThrottleback. If ThrottleBackThreshold is < 1, no throttling back will be performed. When items are subsequently found in the trigger file, the polling interval is set back the original PollInterval property value. . The value of this property defaults to 50.

Class mvDataTable

This class is provided to allow 3rd party controls to be bound to multi-value data via standard .NET databinding interfaces.

The mvDataTable class subclasses the .NET DataTable class – i.e. it inherits from it. Therefore, all the interface members of the DataTable class are supported by this class. However, the mvDataTable class also supports some additional members that are provided to allow an extra degree of control over its usage. Only these extra members are listed in the following table.

An mvDataTable object instance is obtained via the DataTable method of an mvItemList instance; for example:

```
Dim fileProducts As mvFile = myAccount.FileOpen("PRODUCTS")
Dim products As mvItemList = fileProducts.Select()
Dim mvDT As mvDataTable = products.DataTable
```

In fact, the above three lines could be combined into 1 single statement:

```
Dim mvDT As mvDataTable = myAccount.FileOpen("PRODUCTS").Select.DataTable
```

To bind a third party control to an mvDatatTable, simply set the DataSource property of the control to the mvDataTable object:

```
C1FlexGrid1.DataSource = mvDT
```

There are several important things to note about mvDataTable objects. Firstly, the instantiation of an mvDataTable object (via the mvItemList.DataTable method) will force all data in all items within the item list to be retrieved from the server. Secondly, the mvDataTable object has only 1 way of forcing modified rows to be saved to the database - this is via its WriteModifiedRows method. Thirdly, the mvItemList used to instantiate the mvDataTable object is kept in-sync with the modifications made to the mvDataTable.

Method/Property Summary

The mvDataTable class supports the following methods and properties (Type P = Property, M = Method) :

Name	Type	Description
ItemList	P	The parent mvItemList object used to instantiate the mvDataTable object.
WriteModifiedRows	M	Forces all rows that have been modified within the mvDataTable object to be written to the database.

Class mvQueryList

This class allows queries defined using the Data Manager application to be executed within your application code. It presents the results of the query in the form of a collection of rows, each comprising a collection of column values.

The Data Manager chapter contains a section which covers the creation and content of query definitions. Note, the Binding Objects module of mv.NET contains an mvQuery control which can also be used to incorporate queries into your application. Internally, the mvQuery control uses an mvQueryList object.

Property Summary

The mvQuery class supports the following properties:

Name	Description
Row	Allows one row of query data to be retrieved. Returns an mvQueryRow object. See below.
Count	The total number of query rows. The value returned by this property will depend on the HideLevel property setting.

HideLevel	The current level of detail line hiding. The is the subtotaling level below which no detail lines are shown. A value of 0 indicates no detail hiding.
SelectedIndex	The row number currently selected. 1-based index.
Column	Returns a single query column definition in the form of an mvQueryColumn object. See below.
Columns	Returns a collection of mvQueryColumns objects representing all of the columns definitions within a query.
QueryDefinition	The query definition. Returns an mvItem object representing the query definition item from the query definition file.
File	The source of data items for the query. Returns an mvFile object.

The following constituent classes are used in the presentation and storage of query definition and data:

mvQueryRow
mvQueryColumn
mvQueryColumns

These are detailed in the following sections.

Class mvQueryRow

This class allows access to the data on one specific query line. Instances of this class are obtained via the Row property of the mvQueryList class.

Property Summary

The mvQueryRow class supports the following properties:

Name	Description
Value	Allows access to a specific column value of the row via either physical position or column title
BreakLevel	The detail level of this line a query data. Zero indicates bottom level (base) data. Greater than detail indicates than this query line holds subtotaling data.

Class mvQueryColumn

This class allows access to the column definition of one specific column definition within the query. Instances of this class are obtained via the Column or Columns properties of the mvQueryList class

Property Summary

The mvQueryColumn class supports the following properties:

Name	Description
Index	The position of the column within the list of column definitions. 1-based index.
Title	The title of the column.
Width	The default with (in characters) of the column.
Alignment	The data alignment within the column. Returns a DictionaryJustification enumeration: LeftAlign = 0 RightAlign = 1 CenterAlign = 2

Class mvQueryColumns

This class allows access to all of the column definitions within the query.

Property Summary

The mvQueryColumns class supports the following properties:

Name	Description
Item	Returns an mvQueryColumn object representing one specific column definition within the query
Count	The number of columns within the query. 1-based index.

The mvQueryColumns class implements the IEnumerable and IEnumerator interfaces allowing access to its content via For/Each programming constructs.

Class mvSessionControl

This class allows programmatic control of the database session pooling. By obtaining an instance of this class you may query and terminate the current contents of the database session pool.

Method Summary

The mvSessionControl class supports the following methods:

Name	Description
ActiveSessions	Returns an array of mvSession objects representing the current entries within the database session pool.
ForcedRemovalEventEnabled	Use to turn on/off the raising of the ForcedRemoval event. See below.
TerminateSession	Allow a specific entry within the database session pool to be terminated.
TerminateAllSessions	Allow all entries within the database session pool to be terminated.
TerminateAllSessions	Allow all entries within the database session pool to be terminated.

Event Summary

The mvSessionControl class supports the following events:

Name	Description
ForcedRemoval	Raised when a session is forcibly removed from the session pool due to it being detected as inoperative/unresponsive. The setting of the ForcedRemovalEventEnabled property controls whether this event is fired. The arguments of this event provide information relating to the session which has been removed.

Class mvSession

This class represents an entry within the current database session pool.

Property Summary

The mvSession class supports the following properties:

Name	Description
AccountName	The name of the account profile associated with the session.
LastAccessed	A Timestamp indicating when the session was last accessed.
Port	The database process port number associated with the session.
ServerName	The name of the server profile associated with the session.
SessionID	The unique internal identification number of the session.
Status	The current status of the session (Free/Allocated/Shared)

DataBASIC Methods

The BlueFinity.mvNET.CoreObjects.DataBASIC namespace contains a range of methods designed to provide some of the familiar functionality available within MultiValued DataBASIC.

The aim here is not to duplicate every aspect of functionality contained within DataBASIC, but rather to provide those features that are unique to DataBASIC - i.e. those aspects which are not readily available via alternative native methods within .NET.

Method Summary

The following methods are provided:

Name	Description
Char	Returns a character (string) representing the ASCII character at a specified rank position.
Convert	Returns a string where all occurrences of a set of characters have been replaced with an equivalent character from a replacement set.
ConvertString	Returns a string where all occurrences of a set of characters

	have been replaced with an equivalent character from a replacement set.
Count	Returns the number of times a delimiter string occurs in a source string.
Date	Returns the current date as a MultiValued internal date formatted value.
DCount	Returns the number elements of a string that are separated by a delimiter string.
Delete	Deletes an attribute, value or subvalue from within a string in 'item' format.
Extract	Returns an attribute, value or subvalue from within a string in 'item' format.
Field	Returns multi-character delimited substrings from within a string.
IConv	Applies a MultiValued input conversion code to the supplied data.
Index	Returns the position of a character or characters within a string.
Insert	Inserts an attribute, value or subvalue into a string in 'item' format.
Locate	Returns the index of an attribute, value or subvalue with a string in 'item' format.
Num	Returns a Boolean value indicating whether the supplied argument represents a numeric value.
OConv	Applies a MultiValued output conversion code to the supplied data.
Replace	Replaces an attribute, value or subvalue within a string in 'item' format with another value.
Seq	Returns the ASCII rank position of a specific character.
Rnd	Returns a random integer number between zero and the specified argument -1. i.e. the argument indicates the number of possible values for the random number starting from zero.
Space	Returns a string value containing a specified number of space characters.
Str	Returns a string value containing a specified number of occurrences of a specified string.

Time	Returns the current time as a MultiValued internal time formatted string.
Trim	Returns a string with certain space character occurrences removed.

Supported IConv Codes

The following conversion codes are supported by the above IConv function:

D - converts supplied data into an internal date value
MT or **T** - converts supplied data into internal time value
MR or **MD** - scales supplied data into internal integer storage format
X - converts supplied data into hexadecimal form
B - converts supplied data into internal Boolean value

Because MV databases do not support a native Boolean value, the 'B' conversion code must be instructed on what string value represents a Boolean true and false value. The following syntax variations are supported:

B - the `TrueString` static method of the .NET Boolean class is used to control the conversion process.

Bt,f - where t represents the true string value and f the false string value. For example: BY,N or B1,0

Supported OConv Codes

The following conversion codes are supported by the above OConv. The time formatting codes use an example time of 17 minutes and 48 seconds past 11-o-clock in the evening.

DS - converts the supplied internal date value into a short format date string
DL - converts the supplied internal date value into a long format date string
D2 - converts the supplied internal date value into a date string with only a 2-digit year representation
MTSHORT - converts the supplied data into a short format time string
MTLONG or **T** - converts the supplied data into a long format time string
MT - converts the supplied internal time value into a string of format 23:17
MTH - converts the supplied internal time value into a string of format 11:17PM
MTS - converts the supplied internal time value into a string of format 23:17:48
MTSH or **MTHS** - converts the supplied internal time value into a string of format 11:17:48PM
MTSHP or **MTHSP** - converts the supplied internal time value into a string of format PM11:17:48

MR or **MD** – converts the supplied data into a real number representation
X – converts the supplied hexadecimal data into decimal form
B – converts the supplied Boolean data into the either Boolean.TrueString or Boolean FalseString constants. See IConv section above for more details on the syntax variations of this conversion code.

Gateways

This chapter describes mv.NET's flexible remote connectivity feature – the Gateway.

Gateway Overview

There may be occasions where clients require database access from locations where there is no opportunity to establish a direct LAN or VPN connection to the internal database network. In such situations, mv.NET offers a feature whereby such clients can connect via a web service. This web service is known as a 'gateway' or 'gateway service' and is located on a 'gateway server'.

To connect to the database via a gateway service, the client simply needs to specify the URL of the gateway service along with the name of a login profile which is defined in the configuration database being used by the gateway server.

Specifying Access via a Gateway

There are several aspects which need to be configured to allow gateway access. Firstly, the client application needs to use a special overload of the mvEnvironment class in order to specify the URL of the gateway service. Please refer to the Accessing Gateways from the Client section below for further details on this.

Secondly, the gateway service needs to be installed on the gateway server. Please refer to the Installing the Gateway Service section below for further details on this.

Thirdly, the configuration database on the gateway server needs to be defined so that login profile names(s) specified by gateway clients can be recognized by the instance of mv.NET running on the gateway server.

Accessing Gateways from the Client

To access a gateway, the client application must use a specific overload of the mvEnvironment class constructor – this overload allows the URL of the gateway service to be specified. In addition to this URL, a Boolean flag indicating whether the client's database connection is to be persistently reserved for its exclusive use must also be supplied.

Once the mvEnvironment class has been instantiated in this manner, its Login method can be used as normal to obtain a database connection (mvAccount class instance). For example:

```
Dim myEnv As New mvEnvironment(True,
                               "http://mydomain/mvNETGateway/mvNETGateway.asmx")
Dim myAcc As mvAccount = myEnv.Login("SOP")
```

Or, in C#

```
mvEnvironment myEnv = new mvEnvironment(true,
                              @"http://mydomain/mvNETGateway/mvNETGateway.asmx");
mvAccount myAcc = myEnv.Login("SOP");
```

The myAcc variable can then be used in the normal manner in order to access the contents of the account.

Installing the Gateway Service

The CID product comes with a setup routine called 'GWSetup.exe' – this routine needs to be run on each system that is to act as a gateway server. GWSetup.exe installs a similar set of files as per the standard (SRDK setup) server setup routine and should be used instead of this. The main difference between SRDK setup and GWSetup.exe is that the following additional folder is created:

```
C:\Program Files\BlueFinity\mv.NET\Versionx.x\mvNETGateway
```

This folder contains the files and folders which need to be copied into the virtual directory (created by you) to host the Gateway Service.

As mentioned above, an IIS virtual directory needs to be created manually to host the gateway service. This virtual directory can be given any name, but we recommend "mvNETGateway" as a standard. Once the virtual directory has been created, the contents of the mvNETGateway installation folder need copying into the virtual directory folder.

Gateway Hopping

Because the system hosting the gateway service will be accessible by anybody with an internet connection, this system will be typically located within the DMZ area of the network. In order to limit the number and scope of the 'holes' which needs to be opened up through the corporate firewall, it is possible to run 2 gateway services – one within the DMZ (the 'external' gateway) and one on the internal LAN (the 'internal' gateway). This structure means that only port 80 access from the external gateway to the internal gateway system needs to be allowed. Access to the database will be via a transparent 'hop' from the external gateway to the internal LAN gateway.

GWSetup.exe needs to be run on both gateway servers. On the external gateway server, the Data Manager should be used to create a Gateway Profile which points to the internal gateway service. Also, on the external gateway server, all login profiles that are going to be referenced by remote clients should be set to use the appropriate gateway profile.

On the internal gateway server, login profiles should be set up to use server/account profile pairings as normal.

Deploying Your Application

This chapter describes the mv.NET components which you will need to install as part of your own application deployment procedure.

mv.NET's Runtime Deployment Kits

There are 2 versions of runtime deployment kits:

Client Runtime Deployment Kit (CRDK)
Server Runtime Deployment Kit (SRDK)

The SRDK setup file installs all the components installed by the CRDK setup and in addition installs support for the session pooling components, the License Manager and a runtime version of the Data Manager

Using Runtime Deployment Kits

The deployment kits are provided as standalone setup routines that may be executed as part of your own installation procedure. You may run them as a silent background task by supplying a /s option at the end of the invoking command line, e.g.

```
SRDKSetup2.0.exe /s
```

Client Runtime Deployment Kit

This runtime deployment kit installs all of the components necessary to support the client-side libraries of Core Object, Binding Objects and Adapter Objects. It does not install the support libraries for session pooling nor does it install the runtime version of the Data Manager as it assumes that these roles are being supported by a separate system. It intended for use on end-user workstations which are to run interface-only application software, a classic example of this being the deployment of the interface tier of a rich client application.

For rich-client applications, you would typically use the ConfigurationPath feature to allow client systems to access a centrally located configuration database.

Note, also, that the mvEnvironment class has a constructor that allows the ConfigurationPath to be specified programatically.

Server Runtime Deployment Kit

The Server Runtime Deployment Kit should be used on any system which is to support session pooling and or License Management. It installs all the components installed by the CRDK, plus the session pooling components, the License Manager and a runtime version of the Data Manager application. The runtime version of the Data Manager is primarily provided to allow the required server and account profiles to be created and maintained. It will allow a connection to an account to be established but will not allow any activity to be performed within the account, such as viewing the available files and browsing data.

The SRDK is typically installed onto Web servers that are to run mv.NET linked applications or onto servers that are going to run mv.NET session management.

Note, the SRDK does not install any configuration database settings, this must be done by your own installation script according to what configuration settings are required for your application.

Extended Dictionary Deployment

As an integral part of an mv.NET-based application, it is important to make sure that any extended dictionary items that you have created using the Data Manager or other utilities are installed into deployed installations of your application.

All of mv.NET's extended dictionary items are held in the dictionary of the relevant file with item IDs starting with an open curly bracket character ("{") and ending with a closing curly bracket character ("}"). You should make sure that all these items exist in your deployed account(s).

Session Usage Statistics

This chapter describes the session utilization statistics that mv.NET is able to gather and display.

Why Gather Session Statistics?

It may often be difficult to know exactly how large to set your session pools. Sometimes, the only option is to make an educated guess and see what the end user experience is like based on a given setting.

To assist you with this process, mv.NET can gather session utilization statistics for one or more accounts. It also provides a tool which allows you to view these statistics over a period of time.

Activating Statistics Gathering?

On the "Other" tab of the account profile definition is a checkbox which, when ticked, indicates to the Session Manager that when a connection using this account profile is established, it is to start gathering session utilization statistics. This account profile setting is dynamically detected by the Session Manager and so the mv.NET services do not need to be restarted in order for this setting to become active.

Session statistics are gathered and stored on a one-minute interval basis.

Where are Statistics Stored?

All statistical data is gathered into the following folder:

C:\ProgramData\BlueFinity\mv.NET\Version4.0\Statistics

The Statistics for the current Session Manager invocation are held in the "Live" folder under this path. The statistics for previous invocations are held in the Archive folder, which each subfolder within this being given a name indicating the start time of the corresponding invocation of the Session Manager.

When you have finished with statistics you may delete subfolders within the Archive folder as required.

Viewing Session Statistics?

The CID setup and SRDK setup installation routines install the following executable:

mvNET.StatisticsAnalysis.exe

within the bin folder of mv.NET and create a shortcut to the program on the mv.NET Windows Start menu.

A screenshot of the Statistical Analysis tool is show below:

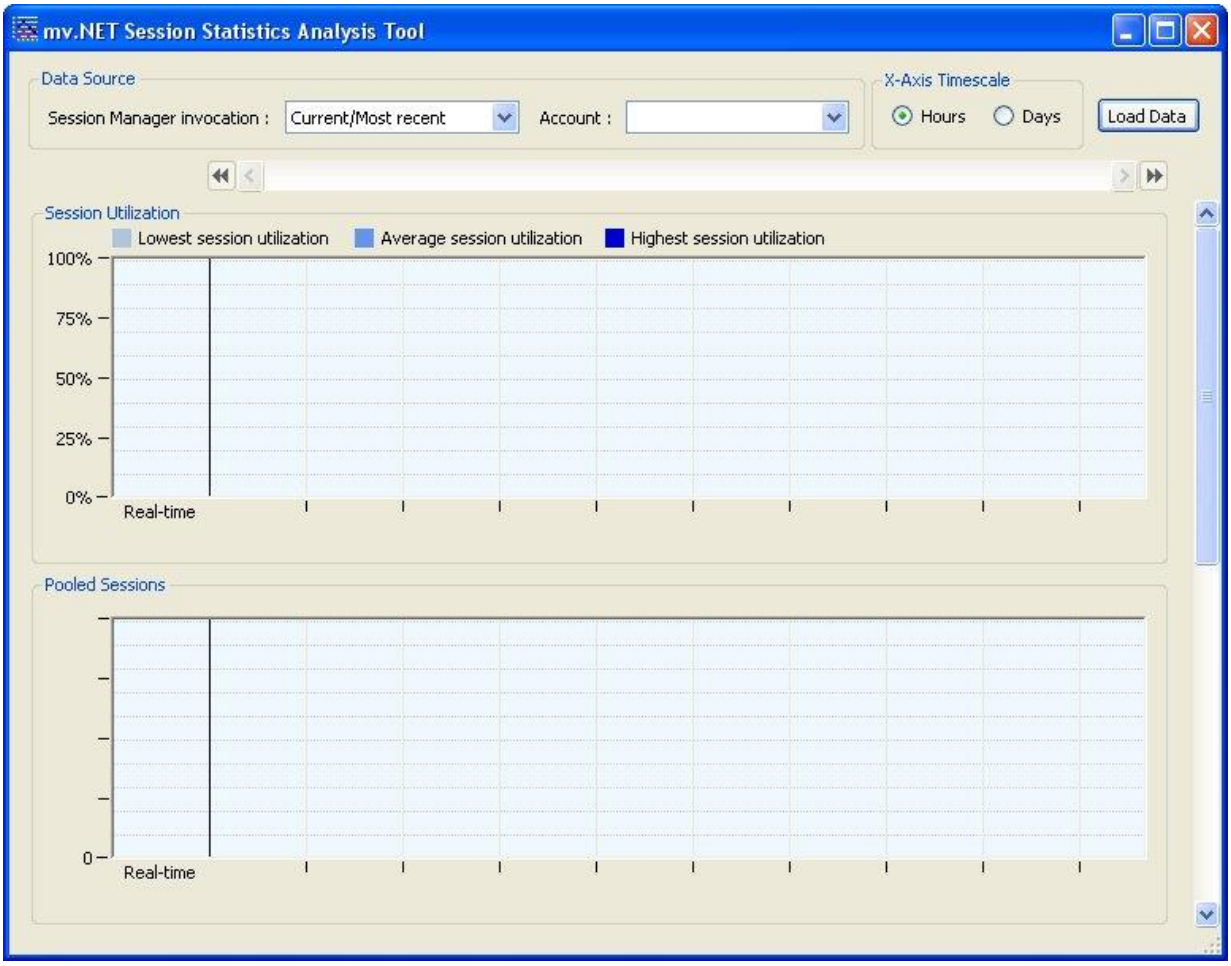


Diagram 10a : The Statistical Analysis Tool's Window

The first combobox at the top of the screen allows you to select the invocation of the Session Manager that is of interest. The dates listed in this combobox relate to the startup date/time of the corresponding Session Manager invocation.

Once you have selected an invocation date/time you are able to select the name of the server/account pairing in which you are interested.

Statistics Category: Session Utilization

The statistics shown in this section of the screen indicate the proportion of time that sessions are in an allocated state as opposed to a free state.

If you are trying to minimize the number of sessions used then if this display indicates an average utilization of around 50% or below, there may be scope for you to reduce the maximum session pool size.

Statistics Category: Pooled Sessions

This statistics grouping allows you to view the number of sessions connected using this server/account profile pairing over time. This is another means to assess whether the maximum size of your session pool is configured in the most appropriate manner.

Statistics Category: Polls taken to Acquire Session

If there are no sessions available at the time a client process requests a database connection from the Session Manager, the client will be asked to wait for several milliseconds (as indicated by the queuing settings within the corresponding account profile) and to then try again. The statistics shown in this grouping allow you to view how many times, on average, clients are being asked to wait each time they request a connection. Obviously, the greater the number here, the greater the delay clients will experience when obtaining sessions. A certain amount of delay is acceptable, but if this number starts to rise above 7 or 8, it may be time to start considering increasing the pool size maximum.

Statistics Category: Session Acquire Requests Per Minute

This statistical grouping allows you to view the number of session requests being generated per minute. This is useful when examining the time profile of session usage by clients.

Troubleshooting

This chapter describes various troubleshooting topics.

Debugging Server-side Code

There are several features available to assist in the task of debugging server-side code.

The Connection Monitor

A window (Connection Monitor) allowing you to view the bi-directional flow of data across a database connection can be viewed by double-click any entry within the Session Monitor's Active Sessions listing. If a server-side routine generates error messages or falls into debug you will be able to see the output in this window. The window will also allow keyboard entry of characters, thus allowing you to debug the server routine in real time

The MVNET.RECORD file

If you need to record the data being passed in and out of the database server over a period of time, you can have mv.NET record this data in a file. You need to create a file called MVNET.RECORD within the data account and create an item called "ACTIVE". The ACTIVE item needs to hold a single integer value:

0 = Recording not active
1 = Record incoming data
2 = Record incoming and outgoing data

The data within the MVNET.RECORD file is held as a series of items with IDs incorporating the port number of connections. Within each item, incoming messages

are proceeded with a line containing "IN =====" ; outgoing messages are proceeded with a line containing "OUT =====".

Note, it is important that recording is deactivated when you have finished your debugging activities as it places a significant overhead on the operation of mv.NET.

The MVNET.TRACE file

By creating a file called MVNET.TRACE within your data account, you will be able to record the last 2 incoming and outgoing messages that occurred on a connection. This feature is designed to provide as feature which can be activated over a period of time when you are trying to identify the last operation to be performed on a connection before the connection was terminated.

The presence of the MVNET.TRACE file is all that is required to activate tracing. Items within this file incorporate the port number of connections, within outgoing messages contained in items containing ".OUT" in their ID and incoming outgoing messages contained in items containing ". IN " in their ID.

The Sample Applications

This chapter describes the sample applications that are installed as part of the CID setup routine.

Application Location

The sample applications are installed into:

C:\Documents and Settings\All Users\Application Data\BlueFinity\mv.NET\Version4.0\Examples

Or, for Vista, Windows7 and Server2008 systems:

C:\ProgramData\BlueFinity\mv.NET\Version4.0\Examples

Core Objects Application

This application illustrates the use of many of the Core Objects classes, namely:

- Accessing the Configuration Database programmatically
- Establishing a database connection
- Opening a data file
- Reading, writing and deleting data items
- Accessing item data
- Selecting items
- Utilizing BTree indexes
- Calling DataBASIC subroutines

Binding Objects Application

This sample application illustrates the use of many of the features within the Binding Objects module.

NOTE, to use this application, you will need to have downloaded the SOP demo account. See section Server Console Window for details on how to do this.

Please refer to the Binding Objects Developer's Introductory Guide for more details of this sample application and Binding Objects in general.

Adapter Objects Application

This sample application illustrates the use of many of the features within the Adapter Objects module. Use of both the visual and non-visual aspects of the ADO.NET data provider are illustrated.

Please refer to the Adapter Objects Developer's Introductory Guide for more details of this sample application and Adapter Objects in general.