MV Share Store Readme

Changes for Version 1.1

Unlimited nested DataPacks as Data Items.

Changes for Version 1.2

MVConfig Library Configuration / Ini File support.

Changes for Version 1.3

New Code field and support for Header / Meta data.

Changes for Version 1.4

DataPack API changed to emulate the standardised C# .NET Collections API (see later).

What is it?

MVShareStore is a programmer's tool implemented as a MS Visual Studio C# Library / DLL that can be used by Windows applications to share data programmatically within a single process and between disparate and separate processes running on the same host.

It can also save this internal or shared data as files to disk which can then be reloaded later if required.

What is included

Fully commented 64 bit and 32 bit .NET C# Libraries / DLL's that can be referenced in any Visual Studio .NET project to implement its data sharing capability. Both Debug and Release versions are available.

The Visual Studio project for a simple Windows Data Server application that can be used to demonstrate how the library can be utilised in your own projects or used as a basis to implement similar capabilities.

The Visual Studio project for a simple Windows Data Client application that can be used to demonstrate how the library can be utilised in your own projects or used as a basis to implement similar capabilities.

The Visual Studio project for a simple Windows Configuration / Ini File handler application that can be used to create, view and manage these files if required.

The downloaded Zip file contains an MVShareStore VS project directory containing the following directories:

bin - All the compiled versions of the MVShareStore Library and the MVConfig and MVXMLParser libraries containing the old legacy API and the new API available.

binnew - All the compiled versions of the MVShareStore Library and the MVConfig and MVXMLParser libraries but with only the new version of the DataPack API available.

doc - All the documentation.

lib - An example directory containing all the libraries needed to be included in your projects.

MVConfigHandler	- The MVConfigHandler example Config program VS Project.
MVDataClient	- The MVDataClient example Client program VS Project.
MVDataServer	- The MVDataServer example Server program VS Project.

Installation

The appropriate (Debug or Release, x64 or x86) MVShareStore.dll library should be copied to a known location where it can be included in your Project's References.

The Library files required are:

MVShareStore.dll	- The Library itself.
MVShareStore.dll.config	- The Library Configuration file.
MVShareStore.pdb	- Library Debug Database.
MVShareStore.xml	- Library Intellisense XML File.

Also required for this release are the following MVConfig support Libraries:

MVConfig.dll	- The MVConfig Library.
MVConfig.pdb	- MVConfig Debug Database.
MVConfig.xml	- MVConfig Library Intellisense XML file.
MVXMLParser.dll	- The MV XML Parser Library.
MVXMLParser.pdb	- MVXMLParser Debug Database.
MVXMLParser.xml	- MVXMLParser Library Intellisense XML file.

The various versions can be found in the following locations in the release ZIP file:

bin\Debug	Debug Any CPU
bin\Release	Release Any CPU
bin\x64\Debug	Debug 64 bit.
bin\x64\Release	Release 64 bit
bin\x86\Debug	Debug 32 bit.
bin\x86\Release	Release 32 bit

These should also be copied to a known location where they can be referenced in your MVConfigHandler project and ideally also located where your MVShareStore Library files are stored so that it can use them.

Please Note:

Your Anti Virus software may complain about the MVShareStore.dll file and try and quarantine it. This is most likely because by nature of its purpose it uses what Microsoft calls 'Unmanaged Code'. It is quite safe however.

- If required the MVDataServer and / or MVDataClient and / or MVConfigHandler projects can be copied to a known location and either referenced, used as a basis or included in your own project(s).
- These projects are in their respective directories in the release ZIP file and can be copied straight to a convenient location and their project files MVDataServer.csproj and MVDataClient.csproj and MVConfigHandler.csproj opened in an existing or new Solution.
- You will then need to delete the existing MVShareStore references in the projects and add a new reference to the MVShareStore.dll you copied above.

- You will also need to add references to the MVConfig.dll and MVXMLParser.dll libraries for this version.
- The project should then compile and run both validating your Library installation and enabling you to manipulate, share and store data.
- You are welcome to use these applications as they are or change them to your hearts content or simply use them as examples of how to utilise this library in your own code.
- Note that these applications come with absolutely no warranty of any sort but suggestions and bug reports (for unmodified versions) are welcome.

Concepts

The MVShareStore Library implements a method of packaging up multiple data items / variables of many different types in a single package (DataPack) that can used internally within an application and / or it can be saved to disk and read back again (DataStore) and / or published into global memory and shared between multiple disparate processes (DataShare).

There are three basic Data containing / handling entities (classes) implemented in this library (plus one additional one now):

DataPack

This is a collection of Data Items of various Data Types such as long, bool and string (see below) and can be thought of as an array containing entries of different data types or a class containing only data.

These Data Items can optionally be named (names must be unique within that DataPack) and Data Items can be queried and found using either their name or their (0 based) index into the 'array'.

It implements the ability to add new entries to it, edit existing entries, insert entries and delete entries, find entries by name or index and cycle through all the entries. DataPacks can also be embedded, merged and sorted.

DataPacks can be used in the following ways:

- Easily converted to a DataStore or DataShare.
- Act as a set of Name / Value pairs
- Act as a composite return value from a function returning multiple variables of different types.
- Act as a composite function parameter containing multiple variables of different types.
- Used as a 'state machine' to communicate between a running thread and the process that owns it.
- Used for any internal communication involving sets of disparate variables.
- Can be implemented as Data Items and nested in other Data Packs to provide sections / subsets of data or act as arrays of Data Items.

DataPack API Changes:

The following changes have been made to the DataPack API designed to bring it more in line with the standardised .NET Collections normal API.

The old legacy API and the new API changes are generally as follows:

Legacy API Calls	**New API Calls**
addDataItem	Add
addDataItems	AddRange
addOrSetDataItem	AddOrSet
setDataItem	Set
getDataItem	GetItem
getBoolValue getIntValue etc.	GetBool GetInt (Value dropped off calls for all supported variable types)
getFirst	GetFirst
hasNext	HasNext
getNext	GetNext
findDataItem	Find
insertDataItem	Insert
insertDataItems	InsertRange
editDataItem	Edit
deleteDataItem	Remove & RemoveAt
clearDataItems	Clear
mergeDataItems	Merge
extractDataPack	Extract
sortDataItems	Sort
getIndex	IndexOf
AllDataItems	GetAllItems

DataStore

This is derived from DataPack and therefore inherits all its capabilities. It implements the ability to save the collections of variables it contains (in the DataPack) to files on disk for future retrieval and re-use.

It can also potentially have other uses such as storing and retrieving program configuration data / settings / state data and almost anything else you can think of.

It can be converted into a DataShare so that it can be shared between processes.

DataShare

This is also derived from DataPack and implements the sharing of these data collections between different processes running on the same host by storing them in RAM using Memory Mapped files.

For example a Server type application may have some data it wants to share with a Client program or two programs might be performing similar tasks and need to communicate to tell each other what they are doing or update each other.

Each DataShare must be named and that name must be unique.

After a DataShare has been created it will have to be posted to the ShareStore (see below) to be made globally available to other applications / processes.

DataShares have the following capabilities:

- A DataShare can have an Owner which can be used to filter what DataShares are returned by searches. This can also optionally be used to group sets of DataShares together.
- They can be read only so that no other processes can modify their contents.
- They can be marked as private so that only the Owner can read and access them (there is no real security provided by the Library here).
- All DataShares are given a unique numeric ID and entered into the Header List which makes them available (by ID or Name) to all other processes using that ShareStore.
- Unless they are marked Private they are also published in a Catalogue so all MVShareStore enabled applications can 'see' them.
- It can be converted into a DataStore so that it can be saved to a file on disk for future re-use.

MVConfigData

This is derived from DataStore and exposes all its functionality and capabilities but also implements some of its own. Its API emulates the MVConfig.MVConfigData API and is used extensively by the new MVConfigFile class.

Please note that Attributes have been included in the API but are not functionally implemented.

This can automatically convert any MVConfig .mvc files it opens into the DataStore compatible .mdc files which it will use thereafter.

ShareStores:

There is the concept of a ShareStore which is a collection containing all the DataShares globally available to any MVShareStore enabled application / process on a host that has opened that ShareStore.

Typically each application will only access a single ShareStore but multiple ShareStores having different names can be implemented on the same host if desired (thereby 'hiding' their data from any application that has a different ShareStore open).

Each ShareStore can contain one or more DataShares which are collections of Data Items (variables) which can be shared between MVShareStore enabled applications.

Normally the default ShareStore will be used and this can be handled automatically by the MVShareStore library itself.

DataTypes:

DataPacks consist of a collection of DataItems (which can optionally be named), have an optional maintenance, flags, section or general purpose field specified and can be of various different DataTypes including:

bool (Bool) - Boolean value
byte (Byte) - Byte (unsigned 8 bit integer) values = 0 - 255
char (Char) - Char (16 bit) values = U+0000 to U+FFFF
short (Short) - Short (signed 16 bit integer) values = -32,768 to 32,767
int (Int) - Int (signed 32 bit integer) values = -2,147,483,648 to 2,147,483,647
long (Long) - Long (signed 64-bit integer) values = -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float (Float) - Float (4 byte numeric) values = ~6-9 digits
double (Double) - Double (8 byte numeric) values = ~15-17 digits
decimal (Decimal) - Decimal (16 byte numeric) values = ~28-29 digits
string (String) - String value
DataPack (Datapack) - Unlimited nested DataPacks

These Data Items can then be accessed using their names (if they were named) or sequentially or by their indexes in the DataPack / DataStore / DataShare collection of Data Items.

How to use it

The Library (MVShareStore.dll) should be fully commented so is self documenting using IntelliSense.

Although this is not its primary function the Library implements some simple forms (as used by the sample Client and Server programs) that can be used for such things as maintaining data, getting input etc. These are usually accessed directly or indirectly through the Data class.

The Library can be used programmatically in two ways:

1. The DataPack, DataShare, DataStore classes can be used and manipulated directly in your code.
2. The DataPack, DataShare, DataStore capabilities can be accessed using the static functions and properties in the Data class (which should never be instantiated).

Data Class capabilities:

Implements all the capabilities of the DataPack, DataStore and DataShare classes through a set of static functions.

Implements the concept of a current DataPack, current DataStore and current DataShare (CurrentDataPack, CurrentDataStore and CurentDataShare properties) which are the default instances which will be used if no other instances of those entities are supplied as a parameter.

Can often optionally display a simple Dialog Box to the user for data input or manipulation. The default is generally not to do this.

The last error or exception that has occurred within the library will be detailed in the LastError property.

The last filename and the last path / folder / directory used by the library are stored in the LastFile and LastPath properties respectively.

Coding Conventions:

- Static Functions in classes like Data and ShareStore are generally capitalised (such as CreateOrOpenShareStore).
- Functions in classes such as DataShare and DataStore are generally camelCase (such as addDataItems).
- DataPack now has a more standardised API based on the normal C# .NET Collections conventions.
- Defines, Types, Structs and Enums are all upper case (such as DataType.TYPE).
- Properties are generally capitalised and named after the Class or Field they represent (such as DataPack or DataShare).
- Many functions have default values defined for some of their parameters (see the IntelliSense comments for these).