

## About Fishhead Software

**URL:**

[www.fishware.com](http://www.fishware.com)

**Email: (for fastest reponse)**

[support@fishware.com](mailto:support@fishware.com)

**Voice:**

630-892-6958

**Fax:**

630-892-6958

**Write:**

Fishhead Software  
900 Ogden Avenue Suite 363  
Downers Grove, IL 60515

## Source Code

**Remember:** You can not distribute the source code or use any portion of it to create commercial, shareware, or freeware ActiveX controls or similar software.

### **Modifying the source code to create an OCX for your company**

Please do the following to the modified control:

- Change the project name and description;
- Change the class name of the control;
- Change the EXE name;
- Change version information to reflect your company;

Note: other people have bought the control and may sell their applications to the same client. Taking these precautions will save you and our clients many headaches.

## System Requirements

**The system requirements are as follows:**

Windows 95 or Windows NT 4.0  
Visual Basic 4.0 (32 bit) or Visual Basic 5.0 or higher  
386 or higher processor  
8 MB RAM  
2 MB disk space

## fsShare ActiveX Demo Version 1.00



[About Fishhead Software](#)

[Copyright](#)

fsShare ActiveX gives the developer an alternative way to communicate to other applications without the use of DDE, clipboard or other COM methods.

### **fsShare5.exe for use with Visual Basic 4.0 and 5.0**

**GUID = C8CD6D7E-4ACD-11D2-A281-000000000000**

**Requires MSVBVM50.DLL**

### **fsShare6.exe for use with Visual Basic 6.0**

**GUID = 15D4F55F-49FD-11D2-A27F-000000000000**

**Requires MSVBVM60.DLL**

[Getting Started](#)

[Frequently Asked Questions](#)

[Features and Uses](#)

[Methods](#)

[Events](#)

[Constants](#)

[System Requirements](#)

**Visit Fishhead Software on the WEB**

[www.fishware.com](http://www.fishware.com)

## Getting Started

fsShare is easy to use. To try it out, we will create two projects: a server and a client.

### Server Application:

1. Create a new project. Select the "standard exe" project type. Name this project Server.
2. From the References dialog, select "Fishhead Software fsShare Demo".
3. Add a form.
4. Now add the following variables to the form's declaration section:

```
Private m_Share As fsShare
Private WithEvents m_Events As fsEvents
Private m_ServerHandle As Long
```

5. Next, in the Form\_Initialize event, add the set statements for the variables define above.

```
Private Sub Form_Initialize()

    Set m_Share = New fsShare
    Set m_Events = m_Share.fsEvents

    m_ServerHandle = m_Share.StartServer("sample server")

End Sub
```

6. Then in the Form\_Terminate event, add the following:

```
Private Sub Form_Terminate()

    m_Share.StopServer

End Sub
```

7. So far we haven't done anything useful, now lets save some data for the client application to use. In the Form\_Load event add the following:

```
Call m_Share.PutData("Company", "Fishhead Software")
Call m_Share.PutData("Product", "fsShare")
```

8. Lets also print out the messages the server application will receive using fsShare. Add the following:

```
Private Sub m_Events_ReceiveMessage(ByVal Handle As Long, ByVal Message As Long, ByVal Data As Variant)

    MsgBox "Handle:" & Str$(Handle) & "    Message:" & Str$( Message) & "    Data:" & Data

End Sub
```

9. Now compile and save this project as Server.exe.

### Client Application:

1. Create a new project. Select the "standard exe" project type. Name this project Client.
2. From the References dialog, select "Fishhead Software fsShare for VB 5" for Visual Basic 4 and 5 applications or select "Fishhead Software fsShare for VB 6" for Visual Basic 6 applications. This must be the same as the one you selected for the Server application.
3. Add a form.
4. Now add the following variables to the form's declaration section:

```
Private m_Share As fsShare
Private WithEvents m_Events As fsEvents
Private m_ServerHandle As Long
Private m_ClientHandle As Long
```

5. Next, in the Form\_Initialize event, add the set statements for the variables define above.

```
Private Sub Form_Initialize()

    Set m_Share = New fsShare
    Set m_Events = m_Share.fsEvents

    m_ClientHandle = m_Share.ConnectClient("sample server", "sample client")
    m_ServerHandle = m_Share.GetServerHandle

End Sub
```

6. Then in the Form\_Terminate event, add the following:

```
Private Sub Form_Terminate()

    m_Share.DisconnectClient

End Sub
```

7. So far we haven't done anything useful, now lets save some data for the client application to use. In the Form\_Load event add the following:

```
' * test to see if the server handle is valid
If m_ServerHandle <> fsSHHNoServerHandle Then
    MsgBox m_Share.GetData("Company")
    MsgBox m_Share.GetData("Product")
End If
```

8. Lets also print out the messages the server application will receive using fsShare. Add the following:

```
Private Sub m_Events_ReceiveMessage(ByVal Handle As Long, ByVal Message As Long, ByVal Data As Variant)

    MsgBox "Handle:" & Str$(Handle) & "    Message:" & Str$( Message) & "    Data:" & Data

End Sub
```

9. Now compile and save this project as Client.exe.

**Try it out**

1. Run the server application.
2. Run the client application. Did the client get the saved data?

## Frequently Asked Questions

- **Can I use fsShare5 with fsShare6 in the same set of applications?**

Yes, but the applications that use fsShare5 will not be able to communicate with applications that use fsShare6. To get all applications to communicate, use fsShare6 or fsShare5, but not both.

- **I am unable to receive events, what is wrong?**

You will need to create three variables, as below, in the declaration section of a form. In the second declaration, notice the use of WithEvents. This tells Visual Basic you want to receive events for this object. Visual Basic will then create two events labeled: [m\\_Events\\_Error](#) and [m\\_Events\\_ReceiveMessage](#).

```
Private m_Share As fsShare
Private WithEvents m_Events As fsEvents
Private m_Handle As Long
```

Next, in the Form\_Initialize event, add the set statements for the variables define above.

```
Private Sub Form_Initialize()

    Set m_Share = New fsShare
    Set m_Events = m_Share.fsEvents

    m_Handle = m_Share.ConnectClient("any server name", "my client")

End Sub
```

- **I tried to install fsShare (fsShare5.exe/fsShare6.exe) onto my clients machine, but when my application runs it gets an error message for inappropriate license file, why?**

This happen because fsShare was not installed correctly. Your installation program will need to install fsShare5.exe and/or fsShare6.exe into the system or system32 sub directory and execute them. fsShare will them update the registry with the correct information.

## Features and Uses

### Features

- Small footprint;
- Uses standard Visual Basic runtime DLL, no additional DLLs or OCXs required;
- Safe and easy to use;
- Allows a system of applications on the same computer to communicate among themselves without the need of DDE, clipboard or other COM methods;
- Handles multiple servers and clients and multiple instances of servers and clients;
- Servers can talk to servers, clients can talk to clients on the same server;
- A system can easily save and/or pass data to other applications within the system;
- An application can fire messages (events) in other applications;

### Uses

- When there is a need for two or more applications to communicate to each other;
- When data needs to be passed to another application and current methods are inadequate;

## Copyright and Trademarks

**COPYRIGHT:** © 1998 Fishhead Software. All Rights Reserved.

fsShare is published under license agreement by Fishhead Software and is protected by United States copyright laws and international treaty provisions.

**TRADEMARKS:** Microsoft and Windows are registered trademarks of Microsoft Corporation. All other brand and product names are trademarks or registered trademarks of their respective holders.

## Constants

Public Enum fsSHFireMessages

fsSHFMServerStarted = 1  
fsSHFMClientStarted = 2  
fsSHFMServerAcknowledge = 3  
fsSHFMClientAcknowledge = 4  
fsSHFMClientUnloading = 398  
fsSHFMServerUnloading = 399  
fsSHFMUser = 400

End Enum

Public Enum fsSHHandle

fsSHHNoServerHandle = -2147480001  
fsSHHNoClientHandle = -2147380001

End Enum

Public Enum fsSHError

fsSHErrNoServerOrClientActive = -2147181503  
fsSHErrServerNotAvailable = -2147181502  
fsSHErrInvalidServerName = -2147181501  
fsSHErrInvalidClientName = -2147181500  
fsSHErrNotValidHandle = -2147181499  
fsSHErrItemNotFound = -2147181498  
fsSHErrInvalidItemName = -2147181497  
fsSHErrItemsReadOnly = -2147181496  
fsSHErrIndexOutOfRange = -2147181495  
fsSHErrNotValidMessage = -2147181494

End Enum

## Methods

[Clear](#)

[ConnectClient](#)

[DisconnectClient](#)

[FireDataMessage](#)

[FireObjectMessage](#)

[fsEvents](#)

[GetClientInfo](#)

[GetData](#)

[GetDataByIndex](#)

[GetIndex](#)

[GetObject](#)

[GetObjectByIndex](#)

[GetServerHandle](#)

[PutData](#)

[PutDataByIndex](#)

[PutObject](#)

[PutObjectByIndex](#)

[SetDefaultServer](#)

[SetReadOnly](#)

[SetReadOnlyByIndex](#)

[StartServer](#)

[StopServer](#)

[Version](#)

## Clear Method

Removes all data associated to the client or server.

### Syntax

object.**Clear**

The Clear method syntax has these parts:

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to fsShare object.

### Remarks

The Clear method, when used by the client, will remove all data items created by the client. When used by the server, the Clear method will remove all the data items created by the server and its clients. When unloading an application, it is not necessary to call this method. fsShare will automatically free the data items. When a client unloads, fsShare will remove the data associated to that client. When the server unloads, fsShare will remove all the data for the server and its clients.

### Example

```
m_Share.Clear
```

## ConnectClient Method

Connects a client application to the server application.

### Syntax

`object.ConnectClient (ByVal ServerName As String, ByVal ClientName As String) As Long`

The ConnectClient method syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to fsShare object;
<i>ServerName</i>	The name of the server to attach to. It must be at least three characters long and leading and trailing spaces will be removed;
<i>ClientName</i>	The name of the client. It must be at least three characters long and leading and trailing spaces will be removed;

### Remarks

The ConnectClient method will connect the client to the named server application. This method will return the client handle.

The ConnectClient method will cause several ReceiveMessage events to be fired. The client will send fsSHFMClientStarted message to the server and to the other attached clients. The server will respond by sending a fsSHFMServerAcknowledge message to the client. The attached clients will respond by sending a fsSHFMClientAcknowledge message to the client. In each case, the handle and the name will be sent with the message.

### Example

```
Private m_ServerHandle As Long
Private m_ClientHandle As Long
Private m_ClientHandles() As Long
Private m_ClientHandlesCount As Long
Private m_Share As fsShare
Private WithEvents m_Events As fsEvents

Private Sub Form_Initialize()

    Set m_Share = New fsShare
    Set m_Events = m_Share.fsEvents

End Sub

Private Sub Form_Load ()

    m_ServerHandle = fsSHHNoServerHandle
    m_ClientHandle = m_Share.ConnectClient("Pacific", "Dolphin")

End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)
```

```
    m_Share.DisconnectClient
```

```
End Sub
```

```
Private Sub m_Events_ReceiveMessage(ByVal Handle As Long, ByVal Message As Long, ByVal Data As Variant)
```

```
    Select Case Message
```

```
        Case fsSHEMServerAcknowledge
```

```
            '** The server will send this message, letting this client know the server's handle  
                m_ServerHandle = Handle
```

```
        Case fsSHEMClientAcknowledge
```

```
            '** Each client will send this message, letting this client know about the other clients  
                m_ClientHandlesCount = m_ClientHandlesCount + 1  
                ReDim Preserve m_ClientHandles(m_ClientHandlesCount) As Long  
                m_ClientHandles(m_ClientHandlesCount) = Handle
```

```
        Case fsSHEMServerStarted
```

```
            '** The server will send this message when it starts, this will only happen when the client starts  
            before the server
```

```
                m_ServerHandle = Handle
```

```
        Case fsSHEMClientStarted
```

```
            '** The cleint will send this message when it starts  
                m_ClientHandlesCount = m_ClientHandlesCount + 1  
                ReDim Preserve m_ClientHandles(m_ClientHandlesCount) As Long  
                m_ClientHandles(m_ClientHandlesCount) = Handle
```

```
        Case fsSHEMServerUnloading
```

```
            '** The server will send this message when it unloads  
                m_ServerHandle = fsSHHNoServerHandle
```

```
    End Select
```

```
End Sub
```

## DisconnectClient Method

Disconnects a client application from the server application.

### Syntax

object.**DisconnectClient**

The DisconnectClient method syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to fsShare object;

### Remarks

This method will disconnect a client from the server application. It is optional to call this method. fsShare will automatically disconnect a client when the client gets unloaded. This method will fire fsSHFMClientUnloading message to all attached clients and to the server.

### Example

```
Private m_ServerHandle As Long
Private m_ClientHandle As Long
Private m_ClientHandles() As Long
Private m_ClientHandlesCount As Long
Private m_Share As fsShare
Private WithEvents m_Events As fsEvents

Private Sub Form_Initialize()

    Set m_Share = New fsShare
    Set m_Events = m_Share.fsEvents

End Sub

Private Sub Form_Load ()

    m_ServerHandle = fsSHHNoServerHandle
    m_ClientHandle = m_Share.ConnectClient("Pacific", "Dolphin")

End Sub

Private Sub Form_Terminate()

    m_Share.DisconnectClient

End Sub
```

## FireDataMessage Method

Fires a message to a client or server passing a variant.

### Syntax

`object.FireDataMessage (ByVal ToHandle As Long, ByVal Message As Long, ByVal Data Variant)`

The FireDataMessage method syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to fsShare object;
<i>ToHandle</i>	An active server or client handle;
<i>Message</i>	The type of message being sent;
<i>Data</i>	The data to be passed to the server or client;

### Remarks

Use the FireDataMessage method to communicate with servers and clients. The ToHandle must be an active server or client handle. The Message must be a value greater than fsSHEMUser. The Data field can be a string, a numeric, any array, a form, a record set, or an ActiveX control. Be aware, when passing an object, it may use its default value. To ensure an object gets passed, use FireObjectMessage instead.

### Predefined Messages

The following message are automatically sent by fsShare:

Message	Value	Method	Triggered	Description
fsSHFMServerStarted	1	StartServer		Sent by a server to unattached clients
fsSHFMClientStarted	2	ConnectClient		Sent by client to server and other attached clients
fsSHFMServerAcknowledge	3	ConnectClient		Sent by server to attaching client
fsSHFMClientAcknowledge	4	ConnectClient		Sent by attached clients to attaching client
fsSHFMClientUnloading	398	DisconnectClient	StartServer	Sent by unattached clients to starting server and other clients
fsSHFMServerUnloading	399	StopServer		Sent by server to other servers and attached clients

Add the following value to your messages (values below this are reserved for fsShare):  
fsSHFMUser = 400

### Example

```
Public Const cCityMessage = 1 + fsSHFMUser    '** User/Application define message
' ** ...
```

Call `m_Share.FireDataMessage(m_ServerHandle, cCityMessage, "Aurora")` ' \*\* tell server the name of the city

## FireObjectMessage Method

Fires a message to a client or server passing an object.

### Syntax

`object.FireObjectMessage (ByVal ToHandle As Long, ByVal Message As Long, ByVal Data As Object)`

The FireObjectMessage method syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to fsShare object;
<i>ToHandle</i>	An active server or client handle;
<i>Message</i>	The type of message being sent;
<i>Data</i>	The data object to be passed to the server or client;

### Remarks

Use the FireObjectMessage method to communicate with servers and clients. The ToHandle must be an active server or client handle. The Message must be a value greater than fsSHEMUser. Unlike FireDataMessage, FireObjectMessage requires the Data field to be an object type. An Object type can a Visual Basic form or ActiveX control.

### Predefined Messages

The following message are automatically sent by fsShare:

Message	Value	Method	Triggered	Description
fsSHFMServerStarted	1	StartServer		Sent by a server to unattached clients
fsSHFMClientStarted	2	ConnectClient		Sent by client to server and other attached clients
fsSHFMServerAcknowledge	3	ConnectClient		Sent by server to attaching client
fsSHFMClientAcknowledge	4	ConnectClient		Sent by attached clients to attaching client
fsSHFMClientUnloading	398	DisconnectClient	StartServer	Sent by unattached clients to starting server
fsSHFMServerUnloading	399	StopServer		Sent by client to server and other clients
				Sent by server to other servers and attached clients

Add the following value to your messages (values below this are reserved for fsShare):

fsSHFMUser = 400

### Example

```
Public Const cCommandButtonMessage = 2 + fsSHFMUser ' ** User/Application define message
' ** ...
Call m_Share.FireObjectMessage(m_ServerHandle, cCommandButtonMessage, Command1) ' ** pass
```

the command button 1 to the server

## fsEvents Method

Allows the application to have events.

### Syntax

```
object.fsEvents () As fsEvents
```

The fsEvents method syntax has these parts:

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to fsShare object;

### Remarks

The fsEvents method allows an application to receive fsShare events. You will need to create a variable with the keyword WithEvents and the set the variable as fsShare.fsEvents type.

### Example

```
Private m_Share As fsShare  
Private WithEvents m_Events As fsEvents
```

```
Private Sub Form_Initialize()
```

```
    Set m_Share = New fsShare  
    Set m_Events = m_Share.fsEvents
```

```
End Sub
```

```
' ** These two events are available when used with WithEvents and Set above
```

```
Private Sub m_Events_Error (ByVal nError As Long, ByVal Description As String, bCancel As Boolean)
```

```
End Sub
```

```
Private Sub m_Events_ReceiveMessage (ByVal Handle As Long, ByVal Message As Long, ByVal Data As Variant)
```

```
End Sub
```

## GetClientInfo Method

The GetClientInfo method returns the number of attached clients and fills in the arrays with client information.

### Syntax

object.**GetClientInfo** (Handles() [As Long](#), Names() [As String](#)) [As Variant](#)

The GetClientInfo method syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to fsShare object;
Handles()	An array of client handles;
Names()	An array of client names;

### Remarks

The GetClientInfo method useful when you need to fire messages to selected group of clients.

### Example

```
Public m_ClientCount As Long  
Public m_ClientHandles() As Long  
Public m_ClientNames() As String
```

```
! ** ...
```

```
m_ClientCount = m_Share.GetClientInfo(m_ClientHandles(), m_ClientNames())
```

## GetData Method

Returns a data item previously saved by PutData or PutObject.

### Syntax

object.**GetData** (ByVal ItemName As String) As Variant

The GetData method syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to fsShare object;
<i>ItemName</i>	The name of a data item;

### Remarks

The GetData method will return a value for the passed in ItemName. In the case GetData cannot find the ItemName, it will return vbEmpty variant data type. When returning an object type, the use of Set is required.

### Example

```
Dim m_EmployeeName As Variant  
m_EmployeeName = m_Share.GetData("Emp_Name")
```

## GetDataByIndex Method

Returns a data item previously saved by PutData.

### Syntax

object.**GetDataByIndex** (ByVal ItemIndex As Long) As Variant

The GetDataByIndex method syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to fsShare object;
<i>ItemIndex</i>	The position within the list of data items;

### Remarks

The GetDataByIndex method will return a value for the passed in ItemIndex. In the case GetDataByIndex is not valid, this method will return vbEmpty variant data type. The GetDataByIndex method can be substantially faster than GetData method. To get the index for an ItemName, save the value returned by PutData.

### Example

```
Dim m_EmployeeName As Variant
```

```
Dim m_EmployeeNameIndex As Long
```

```
m_EmployeeNameIndex = m_Share.PutData("Emp_Name", "John Doe")
```

```
' ** ...
```

```
m_EmployeeName = m_Share.GetDataByIndex(m_EmployeeNameIndex)
```

## GetIndex Method

Returns the data item's index.

### Syntax

object.**GetIndex** (ByVal ItemName As String) As Long

The GetIndex method syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to fsShare object;
<i>ItemName</i>	The name of a data item;

### Remarks

The GetIndex method will return the index for a data item. If an error occurs or the data item is not found, then zero gets return. This method will not generate any trapable errors, making it convient for testing if a data item and server is available.

### Example

```
Dim i As Long
Dim EmployeeName As String

i = GetIndex("Emp_Name")
If i then
    EmployeeName = m_Share.GetDataByIndex(i)
End If
```

## GetObject Method

Returns a data item previously saved by PutObject.

### Syntax

`object.GetObject (ByVal ItemName As String) As Variant`

The GetObject method syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to fsShare object;
<i>ItemName</i>	The name of a data item;

### Remarks

The GetObject method will return a value for the passed in ItemName. In the case GetObject cannot find the ItemName, it will return vbEmpty variant data type. Since GetObject only returns object, the use of Set is required.

To return other types, use GetData.

### Example

```
Dim m_form As Variant
Set m_form = m_Share.GetObject("frmMain")
```

## GetObjectByIndex Method

Returns a data item previously saved by PutObject.

### Syntax

```
object.GetObjectByIndex (ByVal ItemIndex As Long) As Variant
```

The GetObjectByIndex method syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to fsShare object;
<i>ItemIndex</i>	The position within the list of data items;

### Remarks

The GetObjectByIndex method will return an object value for the passed in ItemIndex. In the case GetObjectByIndex is not valid, this method will return vbEmpty variant data type. The GetObjectByIndex method can be substantially faster than GetObject method. To get the index for an ItemName, save the value returned by PutObject. The GetObjectByIndex requires the use of set.

### Example

```
Dim m_Form As Variant
Dim m_FormIndex As Long

m_FormIndex = m_Share.PutObject("frmMain", m_Form)

! ** ...

Set m_Form = m_Share.GetObjectByIndex(m_FormIndex)
```

## GetServerHandle Method

Returns the handle to the connected server.

### Syntax

object.**GetServerHandle** () *As Long*

The GetServerHandle method syntax has these parts:

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to fsShare object;

### Remarks

In order for a client application to communicate with a server application, the client needs to know the server handle.

The GetServerHandle method will retrieve the server handle. Once the server handle been retrieved, the client application can use the FireMessage method to send messages to the server.

### Example

```
Dim ServerHandle As Long
ServerHandle = m_Share.GetServerHandle
If ServerHandle > fsSHHNoServerHandle Then
    ' ** A valid server handle
Else
    ' ** Not attached to a server
End If
```

## PutData Method

The PutData method saves the data into fsShare memory.

### Syntax

`object.PutData (ByVal ItemName As String, ByVal Data As Variant, Optional ByVal Keep As Boolean) As Long`

The PutData method syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to fsShare object;
ItemName	The name for the data being saved;
Data	The value to be saved for later retrieval;
Keep	(Optional) Tells fsShare to keep the data item. (Default: False)

### Remarks

fsShare provides four methods for passing data between applications: PutData, PutObject, FireDataMessage and FireObjectMessage. The PutData and PutObject methods allows an application to save data into fsShare memory area. fsShare will ensure only applications that are attached to the same server can access the data. fsShare allocates separate memory areas for each server, even if they are multiple instances of each other. However, if two servers want to communicate to each other, they can by using the FireDataMessage and FireObjectMessage methods.

Another aspect of PutData is the return value. The return value can be zero, if the ItemName was not added, or the ItemIndex. The returned ItemIndex value can be used by subsequent calls to PutDataByIndex and GetDataByIndex with substantial improvements in performance.

The Keep parameter tells fsShare to keep the data item even after the client has unloaded. By default, fsShare will remove all data items created by the client once the clients gets disconnected. This parameter has no effect for server applications.

### Example

```
Dim ItemIndex As Long
```

```
ItemIndex = m_Share.PutData("Company Name", "Fishhead Software")
```

```
' ** ...
```

```
Dim CompanyName As String
```

```
If ItemIndex > 0 Then
```

```
    CompanyName = m_Share.GetDataByIndex(ItemIndex)
```

```
End If
```

## PutDataByIndex Method

The PutDataByIndex method saves the data into fsShare memory using the ItemName's index.

### Syntax

```
object.PutDataByIndex (ByVal ItemIndex As Long, ByVal Data As Variant) As Long
```

The PutDataByIndex method syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to fsShare object;
ItemIndex	The index into the ItemName list;
Data	The value to be saved for later retrieval;

### Remarks

The PutDataByIndex works the same as PutData, except it uses the ItemIndex instead of the ItemName. Before, PutDataByIndex can be used, the Item must have been saved previously with PutData or PutObject. The return value from PutData and PutObject is the ItemIndex. This value can then be used by PutDataByIndex to improve performance. Like PutData, PutDataByIndex will return zero if the data could not be saved, or the ItemIndex.

### Example

```
Dim ItemIndex As Long
```

```
ItemIndex = m_Share.PutData("Company Name", "Fishhead Software")
```

```
' ** ...
```

```
If ItemIndex > 0 Then
```

```
    ItemIndex = m_Share.PutDataByIndex(ItemIndex, "")
```

```
End If
```

```
' ** ...
```

```
Dim CompanyName As String
```

```
If ItemIndex > 0 Then
```

```
    CompanyName = m_Share.GetDataByIndex(ItemIndex)
```

```
End If
```

## PutObject Method

The PutObject method saves the object into fsShare memory.

### Syntax

```
object.PutData (ByVal ItemName As String, ByVal Data As Object) As Long
```

The PutData method syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to fsShare object;
ItemName	The name for the data being saved;
Data	The object to be saved for later retrieval;

### Remarks

fsShare provides four methods for passing data between applications: PutData, PutObject, FireDataMessage and FireObjectMessage. The PutData and PutObject methods allows an application to save data into fsShare memory area. fsShare will ensure only applications that are attached to the same server can access the data. fsShare allocates separate memory areas for each server, even if they are multiple instances of each other. However, if two servers want to communicate to each other, they can by using the FireDataMessage or FireObjectMessage method.

Another aspect of PutObject is the return value. The return value can be zero, if the ItemName was not added, or the ItemIndex. The returned ItemIndex value can be used by subsequent calls to PutObjectByIndex and GetObjectByIndex with substantial improvements in performance.

### Example

```
Dim ItemIndex As Long
```

```
ItemIndex = m_Share.PutObject("Grid", DBGrid1)
```

```
! ** ...
```

```
Dim grd As Variant
```

```
If ItemIndex > 0 Then
```

```
    Set grd = m_Share.GetObjectByIndex(ItemIndex)
```

```
    MsgBox grd.Row
```

```
End If
```

## PutObjectByIndex Method

The PutObjectByIndex method saves an object into fsShare memory using the ItemName's index.

### Syntax

```
object.PutObjectByIndex (ByVal ItemIndex As Long, ByVal Data As Object) As Long
```

The PutObjectByIndex method syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to fsShare object;
ItemIndex	The index into the ItemName list;
Data	The object to be saved for later retrieval;

### Remarks

The PutObjectByIndex works the same as PutObject, except it uses the ItemIndex instead of the ItemName. Before, PutObjectByIndex can be used, the Item must have been saved previously with PutObject. The return value from PutObject is the ItemIndex. This value can then be used by PutObjectByIndex to improve performance. Like PutObject, PutObjectByIndex will return zero if the data could not be saved, or the ItemIndex.

### Example

```
Dim ItemIndex As Long
```

```
ItemIndex = m_Share.PutObject("Grid", DBGrid1)
```

```
' ** ...
```

```
If ItemIndex > 0 Then
```

```
    ItemIndex = m_Share.PutObjectByIndex(ItemIndex, DBGrid1)
```

```
End If
```

## SetDefaultServer Method

Sets the attached server as the default server.

### Syntax

object.**SetDefaultServer** ()

The SetDefaultServer method syntax has these parts:

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to fsShare object;

### Remarks

In environments where there can be more than one instance of the server application, calling the SetDefaultServer method resolves issues of the client application attaching to the wrong server. By default, the last server load becomes the default server. If there can be only one instance of the server, then there is no need to call this method.

### Example

Call m\_Share.SetDefaultServer

## SetReadOnly Method

The SetReadOnly method sets the ItemName read only flag.

### Syntax

object.**SetReadOnly** (ByVal ItemName As String, ByVal bFlag As Boolean) As Long

The SetReadOnly method syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to fsShare object;
ItemName	The name for the date being saved;
bFlag	A True/False flag for the indicating the ItemName is read only. Default: False;

### Remarks

The SetReadOnly method sets the internal read and write flag for an ItemName. The flag can be set to true or false. When set to True, an application cannot change an ItemName value. When set to False, the default setting, an application can change the ItemName's value at any time.

Another aspect of SetReadOnly is the return value. The return value can be zero, if the ItemName was cannot be found, or the ItemIndex into ItemName list. The returned ItemIndex value can be used by subsequent calls to PutDataByIndex and GetDataByIndex and SetReadOnlyByIndex with substantial improvements in performance.

### Example

```
Dim ItemIndex As Long
```

```
ItemIndex = m_Share.SetReadOnly("Company Name", True)
```

## SetReadOnlyByIndex Method

The SetReadOnlyByIndex method sets the ItemName read only flag using its index.

### Syntax

```
object.SetReadOnlyByIndex (ByVal ItemIndex As Long, ByVal bFlag As Boolean) As Long
```

The SetReadOnlyByIndex method syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to fsShare object;
<i>ItemIndex</i>	The index into the ItemName list;
<i>bFlag</i>	A True/False flag for the indicating the ItemName is read only. Default: False;

### Remarks

The SetReadOnlyByIndex method sets the internal read and write flag for an ItemName. The flag can be set to true or false. When set to True, an application cannot change an ItemName value. When set to False, the default setting, an application can change the ItemName's value at any time.

Another aspect of SetReadOnlyByIndex is the return value. The return value can be zero, if the ItemIndex is not valid, or the ItemIndex into ItemName list. The returned ItemIndex value can be used by subsequent calls to PutDataByIndex and GetDataByIndex and SetReadOnlyByIndex with substantial improvements in performance.

### Example

```
Dim ItemIndex As Long
```

```
If ItemIndex > 0 then
```

```
    ItemIndex = m_Share.SetReadOnlyByIndex(ItemIndex, True)
```

```
End If
```

## StartServer Method

Sets an application as the server application.

### Syntax

object.**StartServer** (ByVal ServerName As String) As Long

The StartServer method syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to fsShare object;
<i>ServerName</i>	The name of the server. It must be at least three characters long and leading and trailing spaces will be removed;

### Remarks

The StartServer method tells fsShare which application is the server application. In each system, one application must be designated as the server application. The server application is typically the first application loaded into memory. Before using GetData or PutData, a server must be active.

The StartServer method fires the fsSHFMServerStarted to all unattached clients and other instances of the server.

### Example

```
Private m_ServerHandle As Long  
m_ServerHandle = m_Share.StartServer("Ocean")
```

## StopServer Method

Tells fsShare this application is no longer the server application.

### Syntax

object.**StopServer**

The StopServer method syntax has these parts:

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to fsShare object;

### Remarks

The StopServer method tells fsShare the application is no longer the server application. After calling this method, subsequent calls to GetData and PutData will fail. The StopServer method will remove all data pertaining to this server.

The StopServer method fires the fsSHFMServerUnloading to all attached clients and other instances of the server.

### Example

Call [m\\_Share.StopServer](#)

## Version Method

Returns the version number for fsShare.

### Syntax

object.**Version** () *As String*

The Version method syntax has these parts:

<b>Part</b>	<b>Description</b>
<i>object</i>	An object expression that evaluates to fsShare object;

### Remarks

The Version method will return a the saved version information in the form of major.minor.revision.

### Example

```
MsgBox m_Share.Version ' Will display 1.00.0000 for the first release
```

## Events

Error

ReceiveMessage

## Error Event

The Error event gets fired when an error occurs.

### Syntax

`Private Sub object_Error (ByVal nError As Long, ByVal Description As String, bCancel As Boolean)`

The Error event syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to fsShare object.
<i>nError</i>	The error number being generated  If <code>nError &gt; 0</code> then ( <a href="#">see Visual Basic help</a> ) else it will be one of the following: fsSHErrNoServerOrClientActive = "No server or client active" = -2147181503 fsSHErrServerNotAvailable = "Server not available" = -2147181502 fsSHErrInvalidServerName = "Invalid server name 'n'" = -2147181501 fsSHErrInvalidClientName = "Invalid client name 'n'" = -2147181500 fsSHErrNotValidHandle = "Not a valid handle" = -2147181499 fsSHErrItemNotFound = "Item 'n' not found" = -2147181498 fsSHErrInvalidItemName = "Invalid item name 'n'" = -2147181497 fsSHErrItemIsReadOnly = "Item 'n' is read only" = -2147181496 fsSHErrIndexOutOfRange = "Index out of range" = -2147181495 fsSHErrNotValidMessage = "Not a valid message" = -2147181494

*Description* A string value representing the generated error.

*bCancel* Determines if an error message dialog displays. When set to [False](#), the default, fsShare will display an error message. When set to [True](#), no message will be displayed by fsShare.

### Remarks

This event is useful if you want to centralize your error handling or prevent fsShare from displaying an error (set `bCancel = True`).

# ReceiveMessage Event

The ReceiveMessage event gets fired when a message is sent to an application .

## Syntax

Private Sub object\_ReceiveMessage (ByVal Handle As Long, ByVal Message As Long, ByVal Data As Variant)

The Error event syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to fsShare object;
<i>Handle</i>	A handle to the calling application;
<i>Message</i>	The message sent by the calling application

## Predefined Messages

The following message are automatically sent by fsShare:

Message	Value	Method	Triggered	Description
fsSHFMServerStarted	1	StartServer		Sent by a server to unattached clients
fsSHFMClientStarted	2	ConnectClient		Sent by client to server and other attached clients
fsSHFMServerAcknowledge	3	ConnectClient		Sent by server to attaching client
fsSHFMClientAcknowledge	4	ConnectClient		Sent by attached clients to attaching client
fsSHFMClientUnloading	398	DisconnectClient		Sent by unattached clients to starting server
fsSHFMServerUnloading	399	StopServer		Sent by server to other servers and attached clients

Add the following value to your messages (values below this are reserved for fsShare):

fsSHFMUser = 400

*Data* A variant representing the data, if any, sent to the receiving application;

