MV.VBX Developer Documentation

What is it and What can it do?

MV.VBX is a Visual Basic Custom Control. When added to your VB project it allows your VB program to respond to **Viewer LDLLHandler Events** previously only accessible to DLL's.

It also provides you with easy access to **Viewer Macro Command Services**. Viewer commands can be issued from the custom control to any instance of Viewer. Full instance information is provided to Visual Basic allowing it to support Multiple Instances of Viewer at the same time.

The control also provides **interrogative services**. It can obtain information about the current state of a Viewer Instance and return title, window handle and color information.

The final feature is **BAGGAGE support**. The control can check baggage for the presence of a file. It can read from the internal file system and allow you to access stored information in baggage. You can use it like a data base or export data on demand to the external file system for use by VB.

To use the VBX, create a VB project and use the Add File command to include the control in your project. Multiple copies of the control may be used within your app as well as control arrays. The control is always invisible at runtime like the timer control.

Viewer Messaging services.

Each time the Viewer Performs a task it sends a message to any DLL it had loaded and in use. These status messages allow the DLL to respond to the Viewer Accordingly. If you minimize the Viewer the DLL is notified and can minimize its windows accordingly.

The problem with this is the Visual Basic cannot directly receive these messages as they can only be received by a DLL. VB has no mechanism to receive them. This means that any Viewer Extension written in VB had to poll for the status of its Viewer window so it knew when to minimize or quit.

The MV.VBX can receive LDLLHandler Messages and convert them into Visual Basic Events. This is done in a two step process. The first stem requires the title that needs VB extensions to register a command inside of a DLL called VBMVLINK.DLL provided with this package. This command is called BroadcastMessages. The following should be placed in the [CONFIG] section of a title wanting to use messaging services.

```
RegisterRoutine("VBMVLINK","BroadcastMessages","U")
BroadcastMessages(hwndApp)
```

Once this is done, any VB apps you launch will receive messages from Viewer if they include the MV.VBX custom control.

Multiple instances of Viewer Can call Broadcast messages and multiple VB apps can listen in to them. Each event received by the control has vwr as the first argument. This argument identified who send the message.

Each running instance of Viewer has a Viewer Instance Handle. This is a unique number assigned to each Viewer Instance. This number is reported as the first parameter of every event so that you can establish who is sending the message.

Also see:

MV.VBX Event Descriptions

Command Services.

The MV.VBX control provides access to the Viewer Macro Command Set. It allows you to direct your commands to the Viewer Instance of your choice. This can be done by either launching a new instance or sending a command directly to a specific running instance of viewer.

There are two ways to direct a viewer command. You must either have an **Vwr value** identifying as specific instance of Viewer or You must have a **title file name**.

To send a command to the Viewer you must:

1. Place a value into Vwr or a Title name in ViewerTitle or provide both. If you with to direct your command directly to a specific instance of Viewer regardless of the title running in it send Vwr and the string value "NULL" as a title name as demonstrated below. If you use the "NULL" value with a valid, unopened MVB file it will be opened for you.

```
' Send a command to a particular Viewer instance
' regardless of the Title running in it
MV1.ViewerTitle="NULL"
MV1.Vwr=1
```

2. Place your Viewer Command in the ViewerCommand Property

```
MV1.ViewerCommand="Next()"
```

3. Set a value for optCmd. If you ignore this property it defaults to 0 which will show the Viewer while your command is executed.

```
MV1.optCmd=0
```

4. To Execute the command set ViewerExecute to True. This will immediately kick off your Viewer command. Remember, you get no advanced about command syntax. The end user sees an unexplainable error message if you submit bad commands.

```
MV1.ViewerExecute=True
```

5. After you execute a Viewer command. The Vwr property gets updated to the value of the Viewer instance that executes your command or 0 if no instance executed your command.

Also See Properties:
optCmd
ViewerCommand
ViewerExecute
ViewerTitle
Vwr

Information Services.

You can inquire about the status, colors and window handles of a Viewer Instance with the information services. This process consists of assigning a Message Code (described below) to VinfoMsg, then you assign a Window Handle in the VGetInfohWnd property if the particular Message requires it as a parameter.

Once you have set these two parameters, set the VGetInfo property to True. This executes your information request, and the results are placed in the VInfo Property.

As an Example, if you needed to know the current Viewer Topic Number of a title you could use the code below.

```
MV1.VInfoMsg=7        ' GI_TOPICNO '
MV1.VGetInfohWnd=0    ' Get the Main Scrolling Region
MV1.VGetInfo=True     ' Kick off the Command
Debug.Print MV1.VInfo ' The results are stored here as a string
```

For a Description of VinfoMsg Values See the VinfoMsg property. Also see VBMVVAR.TXT file for a list of constant names you can include in your VB project.

Also See Properties:
VInfo
VInfoMsg
VGetInfo
VGetInfohWnd

Baggage Services.

Baggage services are a set of features that allow Viewer to store multiple files within its own master mvb file and access them as if they where loose files in a subdirectory. By using baggage services you can store files your program needs within a MVB file for distribution and still access them from visual basic.

The procedure is modeled after VB Binary file access. You indicate a file seek position and buffer size and the control returns the text in a VB string. Keep in mind VB strings have definite size and availability limits. Try to keep your reads down to under 10K blocks.

```
The best way to explain this feature is with an example.
' The following Code Reads a small Baggage file called
' MYLIST.TXT into VB memory and Prints it out in the Debug
' Window. It could easily be modified to export an entire file
' for access by the external file system

MV1.ViewerTitle="MYTITLE.MVB"    ' Provide the name of the title
MV1.BagFile="MYLIST.TXT"         ' Give the name of the baggage file
                                 ' (CASE SENSITIVE)
if MV1.BaggageLOF= 0 Then Goto ExitNoFile  ' Does file exists
MV1.BaggageSeek=0                ' Seek to the start of the file
MV1.BaggageGet=MV1.BaggageLOF    ' Read in the entire file
Debug.Print MV1.BaggageData' Print out the contents of the file
ExitNoFile:
```

Writing to the BagFile property automatically causes a search of the MVB file for the baggage file. If it exists BaggageLOF gets filled with the size of the file. Otherwise it stays at 0. Writing to the BaggageGet Property the amount of bytes required automatically fills BaggageData with the requested data.

Also See Properties:
BaggageFile
BaggageData
BaggageGet
BaggageLOF
BaggageSeek

Licensing

Keyboard Publishing owns and distributes MV.VBX as part of its commercial Viewer products. We have made this product available to developers in the hopes of fostering development with Multimedia Viewer. This is a full, working version of the MV.VBX Viewer Extension.

You are welcome to use MV.VBX in your development and in house projects. If, however you wish to resell or distribute MV.VBX as part of a commercial package a license is required. Please contact Keyboard Publishing at the address listed below.

This product was conceived and developed by **William Reichardt**, an employee of Keyboard Publishing for Multimedia Viewer 1.0 and has been tested and been proven compatible with Viewer 2.0. Keyboard Publishing is not responsible for upgrading or supporting unlicensed customers. Inquiries and questions about this program should be directed via CompuServe Mail to our Technical Support Department.

MV.VBX is © Copyright 1993 by Keyboard Publishing

Tools for teaching, reference & self-study

Keyboard Publishing Inc.
482 Norristown Road, Suite 111
Blue Bell, Pennsylvania 19422
215-832-0945 FAX: 215-832-0948

If you have any further ideas for new capabilities for this extension please drop me a line via CompuServe at 71151,253 or internet 71151.253@COMPUSERVE.COM.

Other Products by Keyboard Publishing.

We are always interested in new ideas for Viewer extension products. If you or your company need custom development work for Viewer extensions please contact us.

We currently have the following products available for Viewer 2.0.

MVScript and The Viewer Commander
A command line/scripting environment for Multimedia Viewer. Allows you to use external script files to store Viewer Macro Commands. The files can later be enclosed in Baggage.

Transcriber
An end user development tool allowing the reader to create cross title custom browse sequences with annotations and save them to external files. Great for students. Will work with any Viewer title.

Score DLL
The score DLL allows you to create Viewer topics that ask questions and keep score for the student providing feedback on learning.

More products are in the works and will be available soon!

MV.VBX Event Descriptions

Events

ActivateWin
EndConfig
EndJump
Init
MinMax
Scroll
Size
StartConfig
StartJump
Term

MV.VBX Property Descriptions

Properties

Activate Event, VBMV Custom Control

Description:
This message is sent when the current Viewer instance is being activated or deactivated.

Syntax:
Sub Activate (vwr **as Integer**,GotFocus **as Long**)

Remarks:
vwr Contains a Viewer Instance Handle.
GotFocus = 0 if Viewer is losing the Focus and 1 if Viewer is gaining focus.

Note: If Viewer is losing the focus then vwr contains the handle of the Viewer instance gaining the focus or 0 if a non Viewer app has gained the focus.

ActivateWin Event, VBMV Custom Control

Description:
This message is sent when a main or secondary window of the current Viewer Instance is being activated or deactivated.

Syntax:

Sub ActivateWin (vwr **as Integer**,WinState **as Long**,WhichWindow **as Long**)

Remarks:
vwr Contains the Viewer Instance Handle
Winstate is 1 if Window is being activated or 0 if being deactivated.
WhichWindow is 1 if a secondary window is be activated or deactivated and 0 if its the Main Window.

ChgFile Event, VBMV Custom Control

Description:
This message occurs when the Viewer is about to load an MVB file. This event can occur under three conditions. When a user loads a new title using file open; By executing an interfile Jump or when a jump to a secondary window is executed.

Syntax:

Sub ChgFile (vwr **as Integer**,MVBFileName**$**, SecondaryWindow **as Long**)

Remarks:
vwr Contains the Viewer Instance Handle
MVBFileName$ contains the name of the New Viewer Title.
Secondary Window =1 if the jump was generated from a secondary window otherwise its 0.

EndConfig Event, VBMV Custom Control

Description:
This event occurs when the Viewer has finished executing the configuration script.
Syntax:

Sub EndConfig (vwr **as Integer**,SecondaryWindow **as Long**)

Remarks:
vwr Contains the Viewer Instance Handle
Secondary Window =1 if the jump was generated from a secondary window otherwise its 0.

EndJump Event, VBMV Custom Control

Description:
This event occurs when a topic jump has occurred. It reports the current topic address after the jump has occurred.

Syntax:

Sub EndJump (vwr **as Integer**,TopicAddress **as Long**,ScrollPosition **as Long**)

Remarks:
vwr Contains the Viewer Instance Handle
TopicAddress contains the Topic address of the topic the jump stopped on.
ScrollPosition indicates the current scroll value of the scroll bar.

Init Event, VBMV Custom Control

Description:
This event occurs when a title first calls the BroadcastMessages Viewer command. If your VB application is not open at this time, it will not see this event.

Syntax:

Sub Init (vwr **as Integer**,hViewerInst **as Long**)

Remarks:
vwr Contains the Viewer Instance Handle
hViewerInst contains the Data Segment Instance Handle of the Viewer. This is very different then a vwr. The hViewerInst can be passed to any Windows API call that requires a true *Instance Handle*.

MinMax Event, VBMV Custom Control

Description:
This event occurs when the main window of the Viewer is Minimized or Maximized.

Syntax:

```
Sub MinMax (vwr as Integer,MinMaxState as Long)
```

Remarks:
vwr Contains the Viewer Instance Handle
MinMaxState = 1 if the Viewer was minimized and 2 if the Viewer was maximized.

Scroll Event, VBMV Custom Control

Description:
This event occurs when the topic has been scrolled.

Syntax:
Sub Scroll (vwr **as Integer**,TopicAddress **as Long**,ScrollPosition **as Long**)

Remarks:
TopicAddress is the topic address of the current scroll position.
ScrollPosition is the position of the scroll bar in the scrolling region.

Size Event, VBMV Custom Control

Description:
This event is sent after the main window is resized.

Syntax:
Sub Size(vwr **as Integer**,iWidth **as Long**,iHeight **as Long**)

Remarks:
iWidth and iHeight contain the dimensions in pixels of the Viewer Window.

StartConfig Event, VBMV Custom Control

Description:
This event occurs when Viewer begins executing the configuration script.

Syntax:

Sub StartConfig (vwr **as Integer**,SecondaryWindow **as Long**)

Remarks:
vwr Contains the Viewer Instance Handle
Secondary Window =1 if the jump was generated from a secondary window otherwise its 0.

StartJump Event, VBMV Custom Control

Description:
This event occurs when the Viewer is about to execute a topic jump. The event reports information about the topic before the jump occurs.

Syntax:

Sub StartJump (vwr **as Integer**,TopicAddress **as Long**,ScrollPosition **as Long**)

Remarks:
vwr Contains the Viewer Instance Handle
TopicAddress contains the Topic address of the topic the jump started on.
ScrollPosition indicates the current scroll value of the scroll bar.

Term Event, VBMV Custom Control

Description:
This event occurs when the current Viewer instance is about to quit. At this time you should clean up any open files and terminate with the title that called it.
Syntax:

Sub Term(vwr **as Integer**,hViewerInst **as Long**)

Remarks:
vwr Contains the Viewer Instance Handle
hViewerInst contains the Data Segment Instance Handle of the Viewer. This is very different then a vwr. The hViewerInst can be passed to any Windows API call that requires a true *Instance Handle*.

BaggageFile Property, VBMV Custom Control

Description:
This is the name of a baggage file you would like to open.

Usage:
[form.]MV1.BaggageFile = NameOfBagFile$

Remarks:
Always set the ViewerTitle property First. Setting this property also sets the BaggageLOF property to the size of the bag file.

Data Type:
String

BaggageData Property, VBMV Custom Control

Description:
This property is used to return data read in from baggage.

Usage:
BagData$=[form.]MV1.BaggageData

Remarks
This property becomes defined by setting the BaggageGet property which causes it to become filled with data.

Data Type:
String (Read Only)

BaggageGet Property, VBMV Custom Control

Description:
When a value X is assigned to this property, X bytes of data is read into the BaggageData Property from the select baggage file pointed to by the BaggageTitle Property..

Usage:
[form.]MV1.BaggageGet = BytesToReadIn

Remarks:
None

Data Type:
Long

BaggageLOF Property, VBMV Custom Control

Description:
Contains the size in bytes of the selected baggage file.

Usage:
SizeOfBagFile=[form.]MV1.BaggageLOF&

Remarks:
This property becomes defined when BaggageTitle is defined.

Data Type:
Long (Read Only)

BaggageSeek Property, VBMV Custom Control

Description:
Moves the file pointer within a baggage file.

Usage:
[form.]MV1.BaggageSeek = BaggageSeekPosition&

Remarks:
This property can be used to move around in a baggage file. Set the BaggageGet property to a size to read in data starting at BagggageSeek.

Data Type:
Long

optCmd Property, VBMV Custom Control

Description:
This property controls how Viewer commands are executed.

Usage:
[form.]MV1.optCmd=Option%

Remarks:
optCmd can take on two values. cmdoptNONE(0) or cmdoptHIDE(1). If cmdoptHIDE is selected your Viewer commands will be performed on the Viewer invisibly. If you create a New Instance of Viewer with this option it will stay hidden until a jump is executed.

Data Type:
Integer

VGetInfo Property, VBMV Custom Control

Description:
This property launches a Viewer info inquiry.

Usage:
[form.]MV1.VGetInfo = True

Remarks:
Assigning this value to True causes the Viewer to respond with the information requested in the VGetInfoMsg property. See this property for definitions of the types of information that will be returned by setting the VGetInfo property.

Data Type:
Integer (WRITE ONLY)

ViewerCommand Property, VBMV Custom Control

Description:
This property contains the Viewer Macro command you wish to execute.

Usage:
[form.]MV1.ViewerCommand = "Next();About();RegisterRoutine('MyDLL','MyFunction','US')"

Remarks:
Multiple Commands can be executed if they are separated by a semicolon.

Data Type:
String

ViewerExecute Property, VBMV Custom Control

Description:
This property issues your Viewer command in the ViewerCommand property.

Usage:
[form.]MV1.ViewerExecute = True

Remarks:
Always Set the vwr, ViewerTitle and cmdOpt properties before setting this value to True to run your command.

Data Type:
Integer (WRITE ONLY)

ViewerTitle Property, VBMV Custom Control

Description:
This property contains the name of the Viewer file you want to send a Viewer command to.

Usage:
[form.]MV1.ViewerTitle = "C:\VIEWER\MYTITLE.MVB"

Remarks:
This property can be set to NULL if you wish to refer to a Viewer title by instance (vwr) only. To do this use the code below:

```
MV1.ViewerTitle = "NULL"
```

Data Type:
String

VInfo Property, VBMV Custom Control

Description:
This property is used to return information from a ViewerGetInfo Call.
Usage:

```
InfoResults$=[form.]MV1.VInfo
wHnd%=Format$([form.]MV1.VInfo)
```

Remarks:
This property get defined only after you set VGetInfo to True; It can contain any message. See VGetInfoMsg property for possible return types.

Data Type:
String (READ ONLY)

VInfoMsg Property, VBMV Custom Control

Description:
This property specifies the information that will be returned in response to setting the VGetInfo property to True.

Usage:
[form.]MV1.BagFile = GI_TOPICNO

Remarks:
This property can take on many enumerated values. See VBMVVAR.TXT for all the GI_ constant values. Some Values may also require you to set the VGetInfohWnd before making a request. See the list given below.

Message	Value	VgetInfohWnd	Return Value
GI_INSTANCE	1	Yes	Integer
GI_MAINHWND	2	No	Integer
GI_CURRHWND	3	Yes	Integer
GI_HFS	4	Yes	Integer
GI_FGCOLOR	5	Yes	Long
GI_BKCOLOR	6	Yes	Long
GI_TOPICNO	7	Yes	Long
GI_HPATH	8	Yes	String
GI_HWNDHELPMAIN	2	??	Integer
GI_HWNDHELP2ND	9	??	Integer
GI_HWNDHELPCUR	10	??	Integer
GI_HWNDSRMAIN	11	No	Integer
GI_HWNDSR2ND	12	No	Integer
GI_HWNDSRCUR	3	No	Integer
GI_HWNDNSRMAIN	13	No	Integer
GI_HWNDNSR2ND	14	No	Integer
GI_HWNDNSRCUR	15	No	Integer
GI_TOPADDR	16	Yes	Long
GI_BOTADDR	17	Yes	Long

For more information about these messages see the Viewer Technical Reference.

Data Type:
String

VGetInfohWnd Property, VBMV Custom Control

Description:
This property serves as a way to pass a parameter to the VGetInfo commands. See VGetInfoMsg property for more information when and how to use this property.

Usage:

```
[form.]MV1.VGetInfohWnd = myWindow.hWnd
```

Remarks:
None
Data Type:
Integer

Vwr Property, VBMV Custom Control

Description:
Viewer instance Handle.

Usage:
[form.]MV1.Vwr = 1

Remarks:
Each time an instance of viewer is created the MVAPI2 DLL assigns it a number to distinguish it from every other Viewer Instance. This is the Vwr Property. Assign this property to the vwr of the Viewer instance you would like to pass a command to.

Data Type:
Integer

Nail Property, VBMV Custom Control

Description:
The window handle of the Viewer Instance your VB App's window should float on top of.

Usage:
[form.]MV1.Nail = VwrhWnd%

Remarks:
If this value is 0 then the App is independent of the Viewer Window. Use VGetInfo commands to obtain a window handle for Viewer. This property allows you to make your VB app look like a real feature of a Viewer Title.

Data Type:
Integer