vList Xtra Help

Installation

What vList Xtra Does

Getting Started

Methods at a Glance

Properties at a Glance

Methods Documentation

Properties Documentation

Shockwave

JavaScript

Using Media of Member with vList

How Director Handles List Sorting

How Director Stores Data Types

Export of AES Encryption

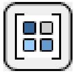
Limitations

How to Order & Register

Licensing & Availability

Technical Support

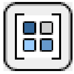
For up-to-date information please visit our web site:
xtras.tabuleiro.com



VLIST XTRA HELP: INSTALLATION

The installation procedure is slightly different depending on the version of Director and platform used. Make sure you have administrative rights to create files in the directory where Director is installed on your system.

WINDOWS	MACINTOSH
Director 11	Director 11
Director MX 2004	Director MX 2004
Director MX	Director MX
Director 8.5	Director 8.5

 VLIST XTRA HELP: INSTALLATION: WINDOWS - DIRECTOR 11

INSTALLING THE XTRA ON WINDOWS - Director 11

Decompress the installation .zip file. This will unpack the Xtra, documentation and sample files to a folder named "vList" on your machine. To install the Xtra, just copy the files Windows\vList.x32 and Windows\vListAuthor.x32 to the Director 11 XTRAS folder. If your copy of Director 11 is installed at the default location, the Windows Xtra files will be located at:

C:\Program Files\Adobe\Adobe Director 11\Configuration\Xtras\vList.x32

C:\Program Files\Adobe\Adobe Director 11\Configuration\Xtras\vListAuthor.x32

Now you need to install the files necessary for creation of cross-platform projector for Mac OSX. Go back to the "vList" directory where the Xtra files were unpacked. Open the **Mac Universal** directory. Now copy the file "vList.cpio" to the "Configuration\Cross Platform Resources\Macintosh\Xtras" directory used by Director 11. In a default installation of Director this file will end up at the following location:

C:\Program Files\Adobe\Adobe Director 11\Configuration\Cross Platform Resources\Macintosh\Xtras\vList.cpio

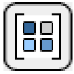
Finally, you need to edit the xtrainfo.txt file to include information about vList. This information is used by the Shockwave and cross-platform publishing features in Director 11, to locate the files needed when assembling the Mac OSX version of your projector. The xtrainfo.txt file is located by default at:

C:\Program Files\Adobe\Adobe Director 11\Configuration\xtrainfo.txt

Double click the file to open it in notepad, or alternatively edit with any other text editor. You need to add the following line to the end of the file:

[#namePPC:"vList", #nameW32:"vList.x32",
#package:"http://download.tabuleiro.com/packages/vList/2/vList"]

You may want to customize this line in the future, to instruct Director to download vList Shockwave packages from the same server that hosts your Shockwave applications. This is covered in more detail at the Using the Xtra in Shockwave section of the documentation. Restart Director for the changes to take effect. The Xtra should be listed when you issue the command "put the xtralist" in the message window. The Xtra functions will also be listed when you click the message window Scripting Xtras button.



VLIST XTRA HELP: INSTALLATION: WINDOWS - DIRECTOR MX 2004

INSTALLING THE XTRA ON WINDOWS - Director MX 2004

If you have not done so, we recommend updating to Director MX 2004 version 10.1 before installing the Xtra. This will allow creation of projectors for Mac Classic and OSX (Director MX 2004 without the update can only create cross platform projectors for OSX.)

Decompress the installation .zip file. This will unpack the Xtra, documentation and sample files to a folder named "vList" on your machine. To install the Xtra, just copy the files Windows\vList.x32 and Windows\vListAuthor.x32 to the Director MX 2004 XTRAS folder. If your copy of Director MX 2004 is installed at the default location, the Windows Xtra files will be located at:

C:\Program Files\Macromedia\Director MX 2004\Configuration\Xtras\vList.x32

C:\Program Files\Macromedia\Director MX 2004\Configuration\Xtras\vListAuthor.x32

Now you need to install the files necessary for creation of cross-platform projector for Mac OSX. Go back to the "vList" directory where the Xtra files were unpacked. Open the **Mac Carbon** directory. Now copy the files "vList.data" and "vList.rsrc" files to the "Configuration\Cross Platform Resources\Macintosh\Xtras" directory used by Director MX 2004. In a default installation of Director these files will end up at the following locations:

C:\Program Files\Macromedia\Director MX 2004\Configuration\Cross Platform Resources\Macintosh\Xtras\vList.data

C:\Program Files\Macromedia\Director MX 2004\Configuration\Cross Platform Resources\Macintosh\Xtras\vList.rsrc

If you are running Director MX 2004 10.1, you can also install the files necessary for creation of cross-platform projector for Mac Classic. Again, go back to the "vList" directory where the Xtra files were unpacked. Open the **Mac Classic** directory. Now copy the files "vList.data" and "vList.rsrc" files to the "Configuration\Cross Platform Resources\Classic\Xtras" directory used by Director

MX 2004. In a default installation of Director these files will end up at the following locations:

C:\Program Files\Macromedia\Director MX 2004\Configuration\Cross Platform Resources\Classic\Xtras\vList.data

C:\Program Files\Macromedia\Director MX 2004\Configuration\Cross Platform Resources\Classic\Xtras\vList.rsrc

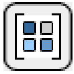
Finally, you need to edit the xtrainfo.txt file to include information about vList. This information is used by the Shockwave and cross-platform publishing features in Director MX 2004, to locate the files needed when assembling the OSX and Classic versions of your projector. The xtrainfo.txt file is located by default at:

C:\Program Files\Macromedia\Director MX 2004\Configuration\xtrainfo.txt

Double click the file to open it in notepad, or alternatively edit with any other text editor. You need to add the following line to the end of the file:

[#namePPC:"vList", #nameW32:"vList.x32", #package:"http://download.tabuleiro.com/packages/vList/2/vList"]

You may want to customize this line in the future, to instruct Director to download vList Shockwave packages from the same server that hosts your Shockwave applications. This is covered in more detail at the Using the Xtra in Shockwave section of the documentation. Restart Director for the changes to take effect. The Xtra should be listed when you issue the command "put the xtralist" in the message window. The Xtra functions will also be listed when you click the message window Scripting Xtras button.



VLIST XTRA HELP: INSTALLATION: WINDOWS - DIRECTOR MX AND 8.5

INSTALLING THE XTRA ON WINDOWS - Director MX and Director 8.5

Decompress the installation .zip file. This will unpack the Xtra, documentation and sample files to a folder named "vList" on your machine. To install the Xtra, just copy the files **Windows\vList.x32** and **Windows\vListAuthor.x32** to the Director 8.5 or Director MX XTRAS folder. If you have previously installed an older copy of the Xtra make sure to remove or replace it.

These are the default locations of the Xtras folder for each application:

Director 8.5- C:\Program Files\Macromedia\Director 8.5\Xtras

Director MX- C:\Program Files\Macromedia\Director MX\Xtras

Finally, you need to edit the xtrainfo.txt file to include information about vList. This information is used by the Shockwave publishing features in Director. The xtrainfo.txt file is located by default at:

Director 8.5 - C:\Program Files\Macromedia\Director 8.5\xtrainfo.txt

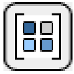
Director MX - C:\Program Files\Macromedia\Director MX\xtrainfo.txt

Double click the file to open it in notepad, or alternatively edit with any other text editor. You need to add the following line to the end of the file:

[#namePPC:"vList", #nameW32:"vList.x32",
#package:"http://download.tabuleiro.com/packages/vList/2/vList"]

You may want to customize this line in the future, to instruct Director to download vList Shockwave packages from the same server that hosts your Shockwave applications. This is covered in more detail at the Using the Xtra in Shockwave section of the documentation. Restart Director for the changes to take effect. The

Xtra should be listed when you issue the command "put the xtralist" in the message window. The Xtra functions will also be listed when you click the message window Scripting Xtras button (Director MX).



VLIST XTRA HELP: INSTALLATION: MACINTOSH - DIRECTOR 11

INSTALLING THE XTRA ON MAC OSX - Director 11

Double-click the installation .dmg file. This will mount a disk named "vList" on your desktop.

The first step is to copy the Universal binaries versions of the Xtra, which will be used in the authoring environment and also when creating Mac OSX projectors, for both Intel and PPC machines. These files are located in the install disk image, at:

vList/Mac Universal/vList.xtra

vList/Mac Universal/vList Author.xtra

These files need to be copied to the Director 11 Xtras folder. The final pathname for the Xtras in a default installation of Director 11 will be:

OSX Volume Name/Applications/Adobe Director 11/Configuration/Xtras/vList.xtra

OSX Volume Name/Applications/Adobe Director 11/Configuration/Xtras/vList Author.xtra

Windows projectors can also be created directly on Director 11 running on Mac OSX after installation of the Windows version of the Xtra. It is located on the install disk, at:

vList/Windows/vList.x32

Copy this file to the Cross Platform resources directory in Director 11, so that it will be available at:

OSX Volume Name/Applications/Adobe Director 11/Configuration/Cross Platform Resources/Windows/Xtras/vList.x32

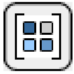
Finally, you need to edit the xtrainfo.txt file to include information about vList. This information is used by the Shockwave and cross-platform publishing features in Director 11, to locate the files needed when assembling the Windows version of your projector. The xtrainfo.txt file is located by default at:

OSX Volume Name/Applications/Adobe Director 11/Configuration/xtrainfo.txt

Double click the file to open it in TextEdit, or alternatively edit with another text editor. Make sure to save the file in plain text format, though. You need to add the following line to the end of the file:

[#namePPC:"vList", #nameW32:"vList.x32",
#package:"http://download.tabuleiro.com/packages/vList/2/vList"]

You may want to customize this line in the future, to instruct Director to download vList Shockwave packages from the same server that hosts your Shockwave applications. This is covered in more detail at the Using the Xtra in Shockwave section of the documentation. Restart Director for the changes to take effect. The Xtra should be listed when you issue the command "put the xtralist" in the message window. The Xtra functions will also be listed when you click the message window Scripting Xtras button.



VLIST XTRA HELP: INSTALLATION: MACINTOSH - DIRECTOR MX 2004

INSTALLING THE XTRA ON MAC OSX - Director MX 2004

Double-click the installation .dmg file. This will mount a disk named "vList" on your desktop.

The first step is to copy the OSX versions of the Xtras, which will be used in the authoring environment and also when creating OSX projectors. These files are located in the install disk image, at:

vList/Mac Carbon/vList

vList/Mac Carbon/vList Author

These files need to be copied to the Director MX 2004 Xtras folder. The final pathname for the OSX Xtras in a default installation of Director MX will be:

OSX Volume Name/Applications/Macromedia Director MX 2004/Configuration/Xtras/vList

OSX Volume Name/Applications/Macromedia Director MX 2004/Configuration/Xtras/vList Author

Director MX 2004 running on Mac OSX can also be used to create Classic projectors, for Mac OS versions 8 and 9. In order to enable this feature you need to copy the Classic version of vList to the correct location in your Director MX installation. First locate the Classic version of vList in the install disk:

vList/Mac Classic/vList

This file needs to be copied to the following location in the Director MX 2004 folder, to be used for cross-platform publishing. Copy it to:

OSX Volume Name/Applications/Macromedia Director MX 2004/Configuration/Cross Platform Resources/Classic MacOS/Xtras/vList

Windows projectors can also be created directly on Director MX 2004 running on Mac OSX after installation of the Windows version of the Xtra. It is located on the install disk, at:

vList/Windows/vList.x32

Copy this file to the Cross Platform resources directory in Director MX 2004, so that it will be available at:

OSX Volume Name/Applications/Macromedia Director MX 2004/Configuration/Cross Platform Resources/Windows/Xtras/vList.x32

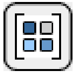
Finally, you need to edit the xtrainfo.txt file to include information about vList. This information is used by the Shockwave and cross-platform publishing features in Director MX 2004, to locate the files needed when assembling the Classic MacOS and Windows versions of your projector. The xtrainfo.txt file is located by default at:

OSX Volume Name/Applications/Macromedia Director MX 2004/Configuration/xtrainfo.txt

Double click the file to open it in TextEdit, or alternatively edit with another text editor. Make sure to save the file in plain text format, though. You need to add the following line to the end of the file:

[#namePPC:"vList", #nameW32:"vList.x32", #package:"http://download.tabuleiro.com/packages/vList/2/vList"]

You may want to customize this line in the future, to instruct Director to download vList Shockwave packages from the same server that hosts your Shockwave applications. This is covered in more detail at the Using the Xtra in Shockwave section of the documentation. Restart Director for the changes to take effect. The Xtra should be listed when you issue the command "put the xtralist" in the message window. The Xtra functions will also be listed when you click the message window Scripting Xtras button.



VLIST XTRA HELP: INSTALLATION: MACINTOSH - DIRECTOR MX

INSTALLING THE XTRA ON MAC OSX - Director MX

Double-click the installation .dmg file. This will mount a disk named "vList" on your desktop.

The first step is to copy the OSX versions of the Xtras, which will be used in the authoring environment and also when creating OSX projectors. These files are located in the install disk image, at:

vList/Mac Carbon/vList

vList/Mac Carbon/vList Author

These files need to be copied to the Director MX Xtras folder. The final pathname for the OSX Xtras in a default installation of Director MX will be:

OSX Volume Name/Applications/Macromedia Director MX/Xtras/vList

OSX Volume Name/Applications/Macromedia Director MX/Xtras/vList Author

Director MX running on Mac OSX can also be used to create Classic projectors, for Mac OS versions 8 and 9. In order to enable this feature you need to copy the Classic version of vList to the correct location in your Director MX installation. First locate the Classic version of vList in the install disk:

vList Folder/Mac Classic/vList

This file needs to be copied to the following location in the Director MX folder:

OSX Volume Name/Applications/Macromedia Director MX/Classic MacOS/Xtras/vList

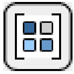
Finally, you need to edit the xtrainfo.txt file to include information about vList. This information is used by the Shockwave and cross-platform publishing features in Director MX 2004, to locate the files needed when assembling the Classic MacOS version of your projector. The xtrainfo.txt file is located by default at:

OSX Volume Name/Applications/Macromedia Director MX/xtrainfo.txt

Double click the file to open it in TextEdit, or alternatively edit with another text editor. Make sure to save the file in plain text format, though. You need to add the following line to the end of the file:

[#namePPC:"vList", #nameW32:"vList.x32", #package:"http://download.tabuleiro.com/packages/vList/2/vList"]

You may want to customize this line in the future, to instruct Director to download vList Shockwave packages from the same server that hosts your Shockwave applications. This is covered in more detail at the Using the Xtra in Shockwave section of the documentation. Restart Director for the changes to take effect. The Xtra should be listed when you issue the command "put the xtralist" in the message window. The Xtra functions will also be listed when you click the message window Scripting Xtras button.

 VLIST XTRA HELP: INSTALLATION: MACINTOSH - DIRECTOR 8.5

INSTALLING THE XTRA ON MAC OS 8 AND 9 - Director 8.5

Running under OSX, double-click the installation .dmg file. This will mount a disk named "vList" on your desktop. To install the Xtra just copy the files "vList" and "vList Author" from the Mac Classic folder to the Xtras folder of your Director 8.5 installation. The final pathname for the Xtras will be for example:

Macintosh HD:OS9 Applications:Macromedia Director 8.5:Xtras:vList

Macintosh HD:OS9 Applications:Macromedia Director 8.5:Xtras:vList Author

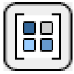
Finally, you need to edit the xtrainfo.txt file to include information about vList. This information is used by the Shockwave publishing features in Director. The xtrainfo.txt file is located in the directory where Director 8.5 was installed, for example at:

Macintosh HD:OS9 Applications:Macromedia Director 8.5:xtrainfo.txt

Double click the file to open it in SimpleText, or another editor capable of saving plain text files. You need to add the following line to the end of the file:

[#namePPC:"vList", #nameW32:"vList.x32",
#package:"http://download.tabuleiro.com/packages/vList/2/vList"]

You may want to customize this line in the future, to instruct Director to download vList Shockwave packages from the same server that hosts your Shockwave applications. This is covered in more detail at the Using the Xtra in Shockwave section of the documentation. Restart Director for the changes to take effect. The Xtra should be listed when you issue the command "put the xtralist" in the message window.



VLIST XTRA HELP: WHAT VLIST XTRA DOES

vList Xtra saves and retrieves Lingo lists and other data types in binary format using either an external file or a cast member for storage. Lists are a powerful and fast way to manage all kinds of information in Director. Since lists can hold Director's rich media and data types they are the natural solution for creating Director databases, yet few developers use them this way. Why? Because saving and retrieving lists at runtime using Director's built-in string() and value() methods is slow and limited.

vList Xtra offers the following advantages over Director's own list saving methods:

RELIABILITY

vList correctly retrieves lists regardless of the data content. Director's value() function is tripped up by certain list content and may not be able to rebuild a list saved as a string.

DATA PROTECTION

vList Xtra offers optional encryption of a stored list using AES (Advanced Encryption Standard) encryption , the standard for secret-key encrypted transfer of unclassified information recently adopted by NIST, the US National Institute of Standards, as a replacement for DES.Encryption can protect sensitive data as well as cast member media that is expensive to produce such as the new 3D cast members in Director 8.5.

COMPRESSION

vList offers optional ZIP compression of stored data. Compression reduces the time necessary to transmit data and decreases the disk space needed.

INTERNET FEATURES

vList can save data to the user's local drive from Shockwave, convert data to Base64 strings for transmission to web servers using postNetText, and it can load vList files from the local hard drive or from remote URL's.

PERFORMANCE

vList, unlike Director's value() function, preserves the sorted state of a saved list when it is retrieved, eliminating the time hit to do a resort upon restoration of the list, and therefore greatly speeding up any subsequent search of the list.

FLEXIBILITY

vList Xtra was designed to store lists, but you can use a vList file or vList member to store any supported data type without first writing it to a list.

```
member("vlist member").content = member(5).media

put member("vlist member").content

-- (media eefe4e0)

member("vlist member").content = "cat"

put member("vlist member").content

-- "cat"

member("vlist member").content = [1,2,3]

put member("vlist member").content

-- [1,2,3]
```

MORE DATA TYPES

vList can read and write the following data types, which are not supported by the Lingo value command:

member.media

member.picture

image (Director 8 and above)

transform (Director 8.5 and above)

void

The media property of a member saved by vList is completely independent of the cast member it came from. You can use this feature to extract members from one movie and recreate them in another.

Data Types Supported by vList Xtra

Lingo Type (ilk)	Example	vList Xtra	Director's Value(listString) Method
#void	void	yes	no
#integer	4 the maxinteger	yes	yes
#symbol	#identif	yes	yes
#string	"some text"	yes (with conversion of high-ASCII characters using Director's fontmap.txt character mapping)	yes (but with no high-ASCII character mapping between Mac and PC)
#object	var = new (script "parent script")	yes	no
#image	image(X, Y, depth) member ().image (the stage).image	yes	no
#picture	member ().picture	yes	no
#float	3.1415926536	yes (full precision, uses new native compact float format when running under Director 8.5)	yes (to the current Lingo float precision)

#list	[1,2,3]	yes	yes
#point	point (1,2) point (1.5, 2.5)	yes, both in integer and float coordinates	yes (float coordinates in the current Lingo float precision)
#rect	rect (1,2,3,4) rect (1.5, 2.5, 5.5, 6.5)	yes, both in integer and float coordinates	yes (float coordinates in the current Lingo float precision)
#propList	[#symb: 2]	yes	yes
#member	member (1,1)	yes	yes
#castLib	castlib 1	yes	yes
#script	member ().script	no	no
#instance	new (script "parent")	yes	no
#xtra	xtra "vList"	no	no
#sprite	sprite 1	yes	yes
#soundsprite	sound (1)	yes yes	yes (Director 7) no (Director 8)
#color	rgb (255, 0, 200)	yes	yes
#date	date(2007,01,01) the systemdate	yes, with limitation	yes
#media	member ().media	yes	no
#window	window movie.dir	no	yes
#vector	vector (1,2,3)	yes (to the full precision)	yes (up to the current Lingo float precision)
#transform	transform()	yes (to the full precision)	no

SIZE

The list size vList can work with is limited only by available memory. Director's value() function is limited to 32,767 entries for a linear list and 16,383 for a property list

BINARY FORMAT

vList saves lists and other data types in their native binary format rather than converting them to and from text. Since no conversion of the data is necessary, vList is up to 20 times faster than Director's value() method, allows lists to be retrieved in a sorted state, and preserves floating point numbers to their full precision.

In addition to saving and retrieving lists and other data types, vList offers several new commands for managing lists.

* aList = concatenate (list1, list2, .., listN)

* sortedState = sortP (aList)

* unSort (aList)

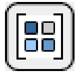
* setPropAt (aPropList, index, property)

* insertAt (aList, index, value)

* insertPropAt (aPropList, index, property, value)

* changeProp(aPropList, oldProp, newProp, deepMode)

* appendProp (aPropList, property, value)



VLIST XTRA HELP: GETTING STARTED

VList operates in two modes, and it can be used both as a Scripting and as an Asset Xtra. Scripting Xtras are used to extend the Lingo language with new functions and datatypes, and vList operates in this mode when it is working with files as its storage medium. But vList can also store its contents directly in vList castmembers (assets), which are saved with your movie and operate like any other built-in Director member type. Each approach has distinct advantages, and we will explain this in more detail in a moment.

However, our first step is to download and install the vList Xtra, following the instructions in the installation page. Now that vList is installed, let's verify that the installation was successful. If you are using DirectorMX or later you should see the vList entry in the Scripting Xtras context menu, appearing at the top of the message window. Selecting the VLIST submenu and the "put interface" entry will output a list of all commands understood by vList in the message window. You can also use the following command

Lingo:

put the xtralist

JavaScript syntax:

trace(_player.xtraList)

to verify which Xtras are installed, including the version number for each one.

We will now try a simple scripting session using vList and the message window. Some of vList functions are available as global Lingo keywords, so there is no need to create an instance of the Xtra in order to use them. For example:

Lingo:

mystring = vList_Version()

put mystring

--"2.0.0"

JavaScript syntax:

var mystring = vList_Version()

trace(myresult)

That's it. You just confirmed that vList Xtra is installed correctly, and its functions are available and ready to be used.

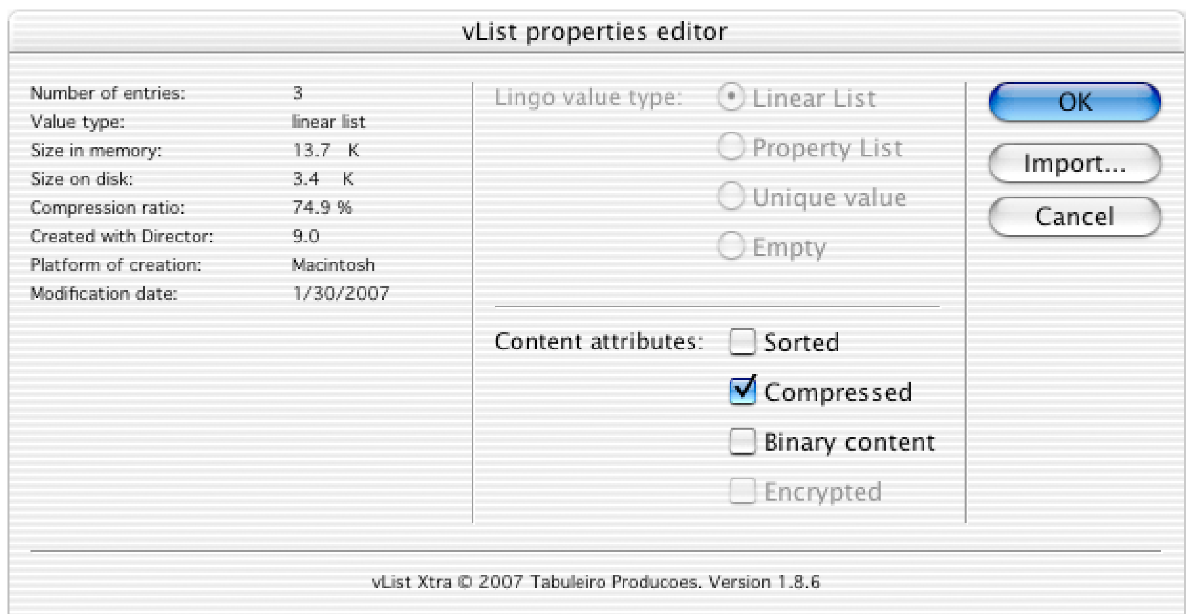
CREATING A NEW VLIST CAST MEMBER

vList Xtra can save a list to an external file or to a vList cast member. You create a vList cast member by doing Insert -> Tabuleiro Xtras -> vList from Director's menu.

You can also create a new member by using the Lingo new member command and passing it type #vlist like so:

memberRef = new(#vList)

When you create a vList cast member by using Insert -> Database -> vList, the following vList member properties dialog appears:



You can view a vList member's properties at any time via this dialog by highlighting the vList member in the cast window, clicking on the Info button at the top of the cast window, and then

clicking on the Options button in the Info Window, or by double-clicking the vList member in the cast window. Items that are dimmed cannot be set from the dialog for that particular member.

Number of entries

Number of items in the list contained by the vList member. 1 if the content is not a list.

Value Type

The type of the data contained by the vList member. Corresponds to the vList member contentType property.

Size in Memory

Size of the data contained by the vList member when loaded into memory. Corresponds to the vList member size property.

Size on Disk

Size of the data contained by the vList member when written out to a file. Corresponds to the vList member sizeOnDisk property.

Compression Ratio

Percentage of disk space saved by compression. 0 if compression has not been activated for the member. Corresponds to the vList member compressionRatio property.

Created with Director

Version of Director the data contained by the vList member was created under. Corresponds to the vList member dirVersion property.

Platform of Creation

Platform (Mac or Windows) the list contained by the vList member was created under. Corresponds to the vList member platform property.

Modification Date

Last date the vList member contents were changed.

Lingo Value Type

The type of data to initialize the vList contents with. This can be changed at any time simply by writing new data to the member. This property is only set-able through the dialog at member creation. After that, the radio buttons show the type of data currently being stored.

Linear - puts [ ] into the member at member creation. Indicates that the member contains a linear list when viewed after member creation.

Property - puts [:] into the member at member creation. Indicates that the member contains a property list when viewed after member creation.

Empty - puts Void into the member at member creation. Indicates that the member contains no value when viewed after member creation.

Unique Value - puts Void into the member at member creation. Indicates that the member contains some Lingo value other than a list when viewed after member creation.

Content Attributes

Sorted - Sorts the member contents and sets the member's sorted flag if pressed. Indicates the current sorted state of the member. Corresponds to the vList member sorted property.

Compressed - Activates compression for the member if pressed. Not available if the member is encrypted because compression has to be applied before encryption. Indicates the current compressed state of the member. Corresponds to the vList member compressed property.

Binary Content - Activates binaryMode for the member. If binaryMode is on, characters in string data will not be cross-mapped when read by vList on the opposite platform (Mac to Win, Win to Mac)

Encrypted - Read-only. Shows whether or not the current contents of the member are encrypted. Cannot be set from the dialog.

Import button

Displays a file picker dialog for choosing a vList list storage file, then imports the contents of the file into the member. Corresponds to the importFile command.

STORING A LIST INSIDE A CAST MEMBER

The most important property of a vList member is its content property. Setting the content property of a vList member to a list in memory writes the list out to the member. Setting a variable to the content property of a vList member puts the list contained by the member into the variable.

```
newMember = new(#vList)

member(newMember).content = [1,2,3]

variable = member(newMember).content

put variable

-- [1,2,3]
```

When you save a movie that has vList members in its castlib, the list data contained by the vList member is saved with the movie. When you save an external castlib with vList members in it, you also preserve their list contents. You can save movies and castlibs on the fly with the following Lingo commands:

```
saveMovie
```

```
save castlib "castlibName
```

When you save a movie or castLib to Shockwave format, the vList members are also compressed using the standard Director compression algorithm, similar to the one used in ZIP archivers.

SAVING A LIST TO A FILE

In order to read from list storage files or write a list out to a file you must first make a file connection to the file by making a new instance of vList xtra and passing it the path to the file, like

so:

```
myFile = new xtra ("vList","C:\TEMP\STORE.LST")
```

The instance variable myFile at this point is linked to file "STORE.LST". You pass it to either the read or write methods to retrieve and store lists to the file like so:

```
myFile = new xtra ("vList","C:\TEMP\STORE.LST")

listVar = [1,2,3]

myFile.write(listVar)

anotherVar = myFile.read()

put anotherVar

-- [1,2,3]
```

REGISTRATION

If you have purchased a registration for the vList Xtra you received a serial number. Make the vList_Register registration call in the first movie that uses vList commands. Make sure you do it before using any other vList commands. The code can be in any type of script.

The startMovie handler is the most convenient place to put it because it will execute before any other code that might try to use vList functions. Remember that a startMovie handler must reside in a movie type of script. You must call the registration command once at the beginning of every Director session where you will be using vList. Once you have registered, the registration stays in memory for the length of the session, even if you go to a movie other than the one that made the registration call.

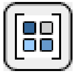
```
on startMovie

vList_Register([111,222,333])

-- notice that the parameter to vList_Register is a list, not a string!
```

end

Previous versions of vList had two levels of registration, Basic and Full, and some Free commands. vList 2 and later do not have these limitations: all functions are available only in the registered version.

LIMITED EVALUATIOON MODE

In unregistered mode most vList functions will cause a warning dialog to display the first time they are used. A limited set of vList utility commands are free to use and do not display a demonstration dialog. Unlike previous versions, all functions that read and store data display a warning when running in trial mode.



VLIST XTRA HELP: METHODS AT A GLANCE

Method and Arguments	Purpose
vList_register (keyList)	Prevents the demo dialog from coming up after purchase.

Saving and Loading Lists

Method and Arguments	Purpose
new xtra ("vList", fileName) new xtra ("vList", fullFilePath) new xtra ("vList", filepathString, | relativepathString |) new xtra ("vList", filepathString, | optionsPropertylist |) new xtra ("vList", filepathString, | singleoptionNumber |)	Returns a vlist instance containing a new file connection for storing or retrieving a list (or any type of data) to a file.
vlistInstance.fileDialogOpen (| optionsPropertylist |)	Displays a file open dialog and returns the path chosen. Sets the file instance to use the chosen path.
vlistInstance.fileDialogSave( )	Displays a file save dialog and returns the path chosen. Sets the file instance to use the chosen path.
new xtra ("vList", urlString)	Returns a vlist instance containing a new file connection for retrieving a list (or any type of data) from a file in the browser cache.
vlistInstance.write (list)	Writes a list (or any type of data) out to a vlist file.

list = vlistInstance.read ( )	Returns the list (or other type of data) stored in a vlist file.
vlistInstance.writeBinary (binaryDatastring, [nullFlag])	Creates a binary file from data stored in a Lingo string
binaryDatastring = vlistInstance.(readBinary [nullFlag])	Reads an entire binary file into a Lingo string
integerLen = lengthBinary (binaryDatastring)	Returns the length of a Lingo string containing binary data
new (#vlist)	Creates a new vList member for storing lists (or any type of data) in the movie's cast
member(vList member).content = list list = member(vList member).content	Setting the content property of a vList member stores a list (or any other data type) inside the member. Setting a variable to the content property of a vList member retrieves the list (or other stored data) and puts it into the variable.

Encryption

vlistInstance.write (list, encryption key)	Writes a list (or any type of data) out to a file.
list = vlistInstance.read (encryption key)	Returns the list (or other type of data) stored in a file.
member(vList member).encrypt (list, encryptionCodeList)	Encrypts and stores the specified list (or other data) in the vList member.
data = member(vList member).decrypt (encryptionCodeList)	Retrieves and decrypts the list (or other data) stored in the vList member
vList_encryptionStrength (integer)	Sets the encryption strenght in bits (64 or 128)

Compression

vListInstance.compression ( ) = true/false	Activates or deactivates compression for the file instance
member (vList member).compression = true/false	Activates or deactivates compression for the member

MIME Base64 Encoding

MIMEencodedString = b64_encode_compress (data,fileName, compressionFlag, | encryptionKey or optionsList |)	Creates a vList file in memory (optionally compressed and/or encrypted) from the passed-in data, then MIME Base64 encodes the vList file and returns it as a string
MIMEencodedString = b64_encode (data,fileName, | encryptionKey or optionsList |)	Creates a vList file in memory (optionally encrypted) from the passed-in data, then MIME Base64 encodes the vList file and returns it as a string.
listData = b64_decode (MIMEencodedString, | encryptionKey or optionsList |)	Extracts vList content (optionally encrypted) from a MIME Base64 string previously created by vList.

Reading and Storing Binary Data

vlistInstance.writeBinary (binaryDatastring)	Creates a binary file from data stored in a Lingo string
binaryDatastring = vlistInstance.readBinary ( )	Reads an entire binary file into a Lingo string
integerLen = lengthBinary (binaryDatastring)	Returns the length of a Lingo string containing binary data
vListInstance.binaryMode (boolean)	Prevents character cross-mapping of strings written to the vlist file or

member("vlist").binaryMode = boolean	member.

File Utility Operations

vlistInstance.fileExist ([encryption key])	Returns 1 if the file linked to the instance exists
vlistInstance.deleteFile ( )	Deletes the file linked to the instance
bytesWritten = vlistInstance.fileSpace ( )	Number of bytes written so far during this session. Useful when running in Shockwave.

vList Member Operations

put data into member (vList member)	Replaces the contents of the vList member with the data passed in, which can be a list or any other data type
Lingo syntax only	
put data after member (vList member)	Appends the data to the contents of the vList member
Lingo syntax only	
put data before member (vList member)	Prepends the data to the contents of the vList member
Lingo syntax only	
member(vList member).importFile (| filePath |)	Imports the list (or other data) contained by the file into the vList member
mediaContents = vList_readFromMedia (vListMediaProperty)	Extracts the data from vList member.media property

boolean = vList_checkMedia (vListMediaProperty)	Does an integrity check on a vList member.media property.

Lingo List Operations

concatenate (list, list, ..., list)	Returns a list containing the elements of the first list followed by the elements of each subsequent list passed in.
insertAt (linearList, positionNumber, anyValue)	Inserts a list item into a linear list at the specified index, and moves the items following it, up one position.
insertPropAt (propertyList,positionNumber, anyProp,`anyValue)	Inserts a property/value pair into a property list at the specified index, and moves the items following it, up one position.
setPropAt (propertyList,positionNumber, anyValue)	Changes the property of the property/value pair at the specified index, but leaves the value untouched.
changeProp (propertyList, oldProp, newProp, deepMode)	Replaces all properties matching oldProp with newProp
appendProp (propertyList, anyProp, anyValue)	Adds a new property/value pair to the end of a property list.
unsort (list)	Removes the internal sort flag from a list but does not change the order of its items.
newList = duplicateSort (list)	Clones the first list into the second list, preserving the sorted state of the original list in the new list.

Creating Populated Lingo Lists

newList = newList (numberOfItems, fillValue)	Creates a new linear list with the specified number of items and fills each item with the specified value. (integer or Lingo symbol only).
newPropList = newPropList (numberOfItems, fillProperty, fillValue)	Creates a new property list with the specified number of property/value pairs and fills each position with the specified property and value. (integer or Lingo symbol only).

Lingo List/Datatype Information

sortP (list)	Returns true if the list is sorted
isSame (variable, variable)	Returns true if the two lists (or any other supported data type) are actually pointers to the same memory location.
numRef (variable)	Returns the number of references to a variable
float32P (float)	Returns true if the passed-in float is a 32-bit float.
float32 (number)	Returns the passed-in value converted to a float that requires only 32 bits for storage.
floatType (float)	Obsolete. Replaced by float32P.
vList_size (list)	Returns the size in bytes of the list (or any other supported data type) when resident in memory (after reading it back from disk).
vList_sizeOnDisk (list)	Returns the size in bytes on disk if the list (or any other supported data type) was

written out to a file.

ShockFiler Xtra Integration

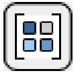
propList = vList_to_sf (data, compressionFlag, | encryptionKey |)	Packages a list or other Lingo value to pass to ShockFiler Xtra
propList = vList_to_sf (data, | optionsList |)	

Miscellaneous

fpReset ( )	Resets the floating point register on Windows
setFileName (memberRef, fileName)	Links a member to an external media file.
vList_version ( )	Returns the release number of the xtra

Error Reporting

vList_error ( )	Returns the number of the last error
vList_errorString (errorNumber)	Returns a text description of the error number.

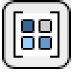
 VLIST XTRA HELP: LIST MEMBER AND FILE PROPERTIES AT A GLANCE

Property	Purpose	Member	File	Get	Set
member(vList member).content = list retrievedList = member(vList member).content	The data contained by the member. Setting the content property of a vList member stores a list (or other data) inside the member. Setting a variable to the content property of a vList member retrieves the list (or other data) and puts it into the variable.	Member		Get	Set
member(vList member).list	Synonym for .content property.	Member		Get	Set
Member syntax: member ("vlist member").dirVersion File syntax: vlistInstance.dirVersion ( )	The version of Director the current data contained by the member or file, was created under.	Member	File	Get	
Member syntax: member ("vlist member").count	Number of items in the list contained by the file or member	Member	File	Get	

File syntax: vlistInstance.count ()				
Member syntax: member ("vlist member").size File syntax: vlistInstance.size ()	Size in bytes of the data contained by the file or member if was loaded into memory	Member	File	Get
Member syntax: member ("vlist member").sizeOnDisk File syntax: vlistInstance.sizeOnDisk ()	Size in bytes on disk of the list contained by the file or member, if it was written out to a file.	Member	File	Get
Member syntax: member ("vlist member").contentType File syntax: vlistInstance.contentType ()	The type of data contained by the member or file. Corresponds to Lingo data type symbols ie #list, #member, #string, #media	Member	File	Get
Member syntax: member ("vlist member").platform File syntax: vlistInstance.platform ()	The platform the current data contained by the member or file was created under.	Member	File	Get
Member syntax:		Member	File	Get

member ("vlist member").sorted File syntax: vlistInstance.sorted ()	The current sort state of the list contained by the member or file				
member ("vlist member").sort = true	Setting the sort property to true sorts the list contained by the member.	Member			Set
Member syntax: member ("vlist member").encrypted File syntax: vlistInstance.encrypted ()	Whether or not the data contained by the member or file is encrypted	Member	File	Get	
vlistInstance.fileName ()	Returns the full filepath and name of the file linked to the vList instance		File	Get	
Member syntax: member ("vlist member").compressed File syntax: vlistInstance.compressed ()	Whether or not the data contained by the member or file is compressed	Member	File	Get	
Member syntax: member ("vlist member").compressionratio	Percent of disk space saved by compression	Member	File	Get	

File syntax: vlistInstance.compressionratio()				
Member syntax: member ("vlist member").binaryContent File syntax: vlistInstance.binaryContent()	Whether or not the string data stored is using character cross-mapping	Member	File	Get



VLIST XTRA HELP: METHODS DOCUMENTATION

vList_register

SAVING AND LOADING LISTS
- newxtra
- fileDialogOpen
- fileDialogSave
- new xtra(urlString)
- write
- read
- new
- content

ENCRYPTION
- write
- read
- encrypt
- decrypt
- encryptionStrength

COMPRESSION
- instance.compression
- member.compression

MIME BASE64 ENCODING
- b64_encode_compress
- b64_encode
- b64_decode

READING AND STORING BINARY DATA
- writeBinary
- readBinary
- lengthBinary
- binaryMode

FILE UTILITY OPERATIONS
- fileExist
- deleteFile
- fileSpace

VLIST MEMBER OPERATIONS
- into
- after
- before
- importFile
- vList_readFromMedia
- vList_checkMedia

LINGO LIST OPERATIONS
- concatenate
- insertAt
- insertPropAt
- setPropAt
- changeProp
- appendProp
- unsort
- duplicateSort

CREATING POPULATED LISTS
- newList
- newPropList

LIST/DATATYPE INFORMATION
- sortP
- isSame
- numRef
- float32P
- float32
- vList_size
- vList_sizeOnDisk

EXPORTING DATA TO SHOCKFILER XTRA
- vList_to_sf

MISCELLANEOUS
- fpReset
- setFileName
- vList_version

ERROR REPORTING
- vList_error
- vList_errorString

vList_register([111,222,333]) - global function, used to register Xtra. It can be called at any time, usually when the Director movie starts. Unregistered versions of the Xtra are functional for evaluation purposes, but will display a warning the first time you use a method other than vList_Register.

vList serial number are strings, and have the generic format VLSXX-1111-2222-3333, where XX is the major Xtra version. In order to protect your serial number from being included as a string in your Director projectors or dcr movies, the vList_Regist function requires only the three groups of numbers 1111, 2222 and 3333 inside a Director list. Leading zeroes do not need to b entered.

An example: if your serial number is VLS20-0123-4567-0089 then you should register using the following command, usually a startmovie handler:

Lingo:

vList_register([123, 4567,89])

JavaScript syntax :

vList_register(list(123, 4567,89))

SAVING AND LOADING LISTS

new xtra("vList",filepathString,|relativePathString|)
new xtra("vList",filepathString,|optionsPropertylist|)
new xtra("vList",filepathString,|singleoptionNumber|) - where filepathString is a string containing either the full path to the fil or just the file name; relativePathString is a string containing either a filename or the full path to the vlist file (Optional argument); optionsPropertylist is a property list containing path information (Optional argument); singleoptionNumber is an integer specifying one numeric option (Optional argument). Returns an instance of vList Xtra linked to the specified file.

Creates a new instance of vList Xtra linked to a file. You must save the vList instance returned by this command into a variabl and pass it as the first argument of read, write, readBinary or writeBinary. This command simply establishes the path to the tar file. If the target file does not yet exist, it will not be created until a write executes.

Simple Case Without Optional Parameters

The simplest use of new is to pass a full file path in filepathString like this:

inst = new xtra ("vlist", "C:\temp\myfile.lst")

A subsequent use of read, readBinary, write or writeBinary will use the specified path to read or create the file. You can also p
just a filename in filepathString.

inst = new xtra ("vlist", "myfile.lst")

In that case, vlist determines the directory to use.

vList automatically adds a file suffix of .LST to the filename passed in, if the filename does not already have a suffix, for compatibility with Windows file naming conventions.

filepathString

You can pass either the full path to the file or just the file name in filepathString. If you pass just a file name, vList constructs
path to the file when it is time to read or write as follows:

vList File Command	Authoring, Projector	Shockwave
write, read	same directory as movie (Lingo's the moviepath)	folder named "vLists" inside the Shockwave support folder
writeBinary, readBinary	same directory as movie (Lingo's the moviepath)	folder named "dswMedia" inside the Shockwave support folder

If the movie has not been saved yet so there is no moviePath, Director's authoring directory is used as the base. In Shockwave
subfolder called "vLists" or "dswMedia" inside the Shockwave support folder on the user's boot drive is used as the base. The
vLists folder is created by vList Xtra. The dswMedia folder is Macromedia's standard folder for restricted reading and writing
media files in Shockwave.

Macromedia has changed the location and name of the DSWMEDIA folder several times. Since there is no function in the Xtr
API to retrieve the path, it is up to Xtra developers to hardcode the changes into their xtras. You may notice on OSX that vLis
uses this path for Shockwave 10.0

HD:Library:Application Support:Macromedia:Shockwave 10:DswMedia

and this one for Shockwave 10.1

HD:Users:User1:Library:Application Support:Macromedia:Shockwave

10:DswMedia

and this one for Shockwave 11

HD:Users:User1:Library:Application Support:Adobe:Shockwave

11:DswMedia

That is because Macromedia and Adobe changed the DSWMEDIA path between those 3 releases.

Protected Web Media

One way to make use of read/writeBinary in Shockwave is to retrieve needed media at runtime. Doing the following in Lingo links the cast member to a file named "image.jpg" in the user's local dswmedia folder:

member("image").filename = "image.jpg"

-- this Lingo command is currently broken on Mac in Shockwave but there is a workaround

The following code extracts an image file from a remote vList container and links to it temporaily. When it is no longer neede can be deleted.

preloadNetThing("http://www.mydomain.com/vListFile.lst")

...

inst = xtra ("vList").new ("http://www.mydomain.com/vListFile.lst")

-- Decrypt content, which is a property list that looks like

-- [#FlashFile1: "dj+(&^jd883638..", #FlashFile2: "3bx638..." ...]

propertyList = inst.read ([33, 44, 55, 22, 44, 66, 77])

binDataForOneFile = propertyList.FlashFile1

-- establish a path in dswMedia

inst = xtra ("vList").new ("flashfile1.swf")

inst.writeBinary (binDataForOneFile)

-- link the file in dswMedia to a cast member

member ("FlashFile1").filename = "flashfile1.swf"

....

-- All done. Get rid of file.

member ("FlashFile1").filename = empty

inst.deleteFile()

|relativePathString|

If you pass just a file name, you can use the optional parameter relativePathString to specify a path in subfolders below the current movie's path. If you pass a full filepath in filepathString, relativePathString is ignored. If you pass a single or nested folder path in relativePathString and the folder does not exist, vList will create the folder or folder hierarchy when it creates th file.

You can use the @ sign instead of the platform-specific path separators, in a relative path. The following relative path when issued from a projector:

new (xtra "vlist", "datafile.lst", "Students@Morning")

will translate into one of these depending whether it was running on Mac or PC

[the moviePath]\Students\Morning\datafile.lst

[the moviePath]:Students:Morning:datafile.lst

When issued in Shockwave and used with the read command it will translate to:

[Shockwave support folder]\vLists\Students\Morning\datafile.lst

When issued in Shockwave and used with the readBinary command it will translate to:

[Shockwave support folder]dswmedia\Students\Morning\datafile.lst

Note: vList unfortunately has its own use for the @ sign that has nothing to do with Lingo's use of it to specify relative paths. this vList context @ is only a substitute for \ or :. If vList sees @ at the beginning of a relative path string, it will try to resolve as a system path, as described in the next section.

vList does not impose any limit beyond what Director and the OS do on the length of the file name or entire file path. Limits cross-platform are 31 characters for file name including extension and 255 characters for the full file path including the file na

System Folder Key Words

You can specify a system folder as the starting place for the relative path string by using a vList keyword at the beginning of t path. To write to one of the pre-defined system folders you start the relative path with the @ sign and the vList keyword for th folder you want to access. For instance, the following opens an instance of vList that will access the file at path C:\WINDOWS\SYSTEM\TEMP\SUBDATA.LST

inst = new xtra ("vlist","data.lst","@TMPDIR\SUBFOLDER\DATA.LST")

Keyword	Win 95	Win 98	class="colorvlist"Win NT	Win ME	Win 2000,XP	Mac Classic	Ma
@SYSPRF	C:\ WINDOWS\ xtrasdir\	C:\ WINDOWS\ xtrasdir\	C:\ WINDOWS\ xtrasdir\	C:\ WINDOWS\ All Users\ Documents \	C: \ Documents and Settings \ All Users \ Documents \	BootDrive : System Folder : Preferences:	Bo Li Pr
@USRPRF	C:\ WINDOWS\ xtrasdir\	C:\ WINDOWS\ Application Data \	C:\ WINDOWS\ Profiles \ [username] \ Application Data \	C:\ WINDOWS\ Application Data \	C: \ Documents and Settings \ [username] \ Application Data\	BootDrive : System Folder : Preferences:	Bo Us [us Li Pr
@TMPDIR	C:\ WINDOWS \ TEMP\	C:\ WINDOWS \ TEMP\	C: \ TEMP \	C:\ WINDOWS \ TEMP\	C: \ Documents and Settings \ [username] \ Local Settings\ Temp \	BootDrive : Temporary Items:	Bo pri 50 Te Ite

This method of accessing system folders is limited, but the key words and usage are easy to remember. Version 1.6.5 introduc a more comprehensive alternate method that uses system folder type numbers to access many more folders.

You can use the @ sign in a system-defined folder path to make it work cross-platform:

@TMPDIR@SUBFOLDER@DATA.LST

You must start a system-defined folder relative path with an @ sign and the keyword in upper case, otherwise the key word w not be recognized.

Note: On systems like XP and OSX the user may not have permission to write to some system directories. In that case vList returns an error and cannot complete the write.

Shockwave

When running under Shockwave, you cannot use a full path for the filepathString argument. You can, however, use the relativ path arguments to create subdirectories in the Shockwave support folder or you can use the system-defined folders to save to those locations from Shockwave.

Examples:

fileLink = new xtra ("vList","C:\TEMP\STORE.LST")

-- Creates a link to a file at the absolute path "C:\TEMP\STORE.LST"

fileLink = new xtra ("vList","INFOFILE")

-- Creates a link to file "INFOFILE.LST" in the same directory as the movie if running from projector

-- Creates a link to file "INFOFILE.LST" in folder "vLists" at the root of the Shockwave support

-- folder if running in Shockwave

fileLink = new xtra ("vList","FOO.LST","DATA")

-- Creates a link to file "FOO.LST" in the directory "DATA", which is a

-- subdirectory of the movie's directory.

fileLink = new xtra ("vList","Phonics","Lessons@English")

-- Creates a link to file "Phonics.LST" in the directory "English", which is a

-- subdirectory of "Lessons", which is a subdirectory of the movie's directory.

-- The path "Lessons:English" will only work on both platforms because it uses

-- the cross-platform path delimiter character.

|optionsPropertylist|

The options property list is a list of option names and their values. Since the options are named it doesn't matter what order the are passed in. The value of the #type option determines what other options can be specified and how they are interpreted.

Folder Type Options

[#type: foldertypeNumber, #domain: osxDomain, #location: pathRelativeToFolder]

foldertypeNumber - Both the Windows and Mac System programming APIs have many predetermined directory paths that ca be referenced from C code using a 4-byte value for the directory. For instance the Desktop folder and the current user's Documents folder have codes. vList can accept this value as an integer and pass it on to the system to allow access to any foldertype recognized by the API. There are hundreds of folder types on both platforms. Rather than duplicate them in vList d the Director movie prefs.dir contains the lists for both platforms and a script for easily accessing them by symbol or by numbe

You can find Microsoft documentation for folder types here.

Apple doc for folder types is not so consolidated. These URL's are the place to start:

constant_6.html

APIIndex.html

Note that Win doc represents the type code in hex and Mac doc represents it as 4 characters. The movie prefs.dir has handlers that you can use to convert either format into the integer representation that vList needs for the foldertype value.

domain - Optional parameter. A domain is an OSX-only classification above folder type that determines that path to the folde type. Currently there are System, Local, User, Network and Classic domains. Domains codes are also referenced with a large integer in vList. The movie prefs.dir provides a list of domain types and code for finding the right number to use to reference a domain.

Here is an example of how the same folder type points to two different paths depending on which domain is specified:

Foldertype: kFontsFolderType (1718578804)

Domain: kUserDomain (-32763)

BootDrive: Users:username: Library: Fonts:

Domain: kSystemDomain (-32766)

BootDrive: System: Libarary: Fonts:

You can learn more about domains here:

constant_11.html

location - Optional parameter. String containing a folder path relative to the folder specified by folder type and possibly doma

Example:

-- kApplicationsFoldertype = 1634758771]

--kSystemDomain = -32766

inst = new xtra("vlist","file.lst",[#type:1634758771 ,#domain: -32766, #location:"MyApp:Data"])

The type and domain params together specify the path :

BootDrive: Applications:

The location param specifies a subdirectory, which will be created if it doesn't exist, relative to the initial directory:

BootDrive: Applications: MyApp: Data: file.lst

Open/Save Dialog Options

[#type: -1, #prompt: dialogText, #location: initialDirectory]

type - Passing a type of -1 causes an open file dialog to display when read or readBinary are used with the instance, and cause save file dialog to display when write or writeBinary are used with the instance. The path to the file is obtained at this time fro the user. If the user Cancels the dialog then the read or write op will return a "User access denied" error.

prompt - Optional param. String with text to display on the dialog like "Choose your scores file"

location - Optional param. In this context the location param does not contain a relative path. It contains the full path to the ini directory the dialog should show the user, or a folder type number for the initial directory. On Mac Classic, using #location to specify the start directory for the dialog is supported in System 8.5 and above.

Examples:

inst = xtra ("vList").new ("aFileName", [#type:-1])

inst = xtra ("vList").new ("aFileName", [#type:-1, #prompt: "a useful description"])

inst = xtra ("vList").new ("aFileName", [#type:-1, #prompt: "Choose your Music Prefs file", #location: 837474747])

inst = xtra ("vList").new ("aFileName", [#type:-1, #prompt: "Choose a folder to save your sounds", #location: "C:\dir1\dir2"])

Local Shockwave Playback Options

[#type: -2, #location: relativePath]

type - Passing a type of -2 causes vList to use "the moviepath" as a starting point when a Shockwave movie is located on the user's hard drive and playing back through Shockwave. Since you cannot build a path using "the moviepath" under Shockwav

with Lingo, this feature allows you to save and read vList files and binary files relative to a Shockwave movie located on the user's hard drive. It does not allow you to save or read files relative to a Shockwave movie that resides on a web server.

location - Optional param. String containing a folder path relative to the local Shockwave movie's path.

Example

-- Shockwave movie is at path "Macintosh HD:testing: movie.dcr"

inst = new xtra("vlist","local.jpg",[#type: -, #location:"graphics"]

-- The path to this file is "Macintosh HD:testing: graphics: local.jpg"

data = inst.readBinary()

|singleOptionNumber|

If you only need to specify the type param without any additional options, you can pass the type number as the third paramete new without creating an options property list. The value for type can be a folder type number, -2 or -1.

Examples

inst = new xtra("vlist","prefs.lst",1886545254) -- kPreferencesFoldertype

-- Creates a path to file prefs.lst in the System Preferences folder

inst = new xtra("vlist","local.lst",-2)

-- Creates a path to file local.lst in the same folder as the local Shockwave movie

inst = new xtra("vlist","test.lst",-1)

-- Prompts the user for the path to "test.lst" when read, readBinary, write or writeBinary are used with this instance.

vlistInstance.fileDialogOpen(|optionsPropertylist|) - where optionsPropertylist is a property list for customizing the dialog. Optional argument. Returns the filepath chosen in the dialog, or "" if no path was chosen. Always returns "" in Shockwave.

This command returns the filepath chosen and also resets the filepath of the file instance to the chosen path. The string you pa in for filename when you create the instance will be overwritten with the path chosen or "" if the user cancelled the dialog. Do

not pass a full file path in the instance if you are going to use file dialogs. They will not open if the existing path contains file separators (: or \ ). In Shockwave this command allows you to work with a file the user has chosen, but you will not have acc to the actual file path.

The optionsPropertyList is the same format used for the new command, except you do not have to pass the #type property, wh is ignored.

Note: You should check the return value of this method or check vList_Error in case the user pressed Cancel, so you can discontinue further operations. If you call a read method for the instance after displaying a dialog where the user pressed Canc there is still no valid filepath, so the read method will display the Open dialog again.

vlistInstance.fileDialogSave(|optionsPropertylist|) - where optionsPropertylist is a property list for customizing the dialog. Optional argument. Returns the filepath chosen in the dialog, or "" if no path was chosen. Always returns "" in Shockwave.

This command returns the filepath chosen and also resets the filepath of the file instance to the chosen path. The string you pa in for filename when you create the instance is used for the default save file name in the dialog. Do not pass a full file path in instance if you are going to use file dialogs. They will not open if the existing path contains file separators (: or \ ). In Shockwave this command allows you to work with a file the user has chosen, but you will not have access to the actual file pa

The optionsPropertyList is the same format used for the new command, except you do not have to pass the #type property, wh is ignored.

Note: You should check the return value of this method or check vList_Error in case the user pressed Cancel, so you can discontinue further operations. If you call a write method for the instance after displaying a dialog where the user pressed Can there is still no valid filepath, so the write method will display the Save dialog again.

newxtra("vList",urlString,) - where urlString is a string containing the URL for a file previously retrieved using the Lingo command preloadnetthing. Returns an instance of vList Xtra linked to the specified file in the internet cache.

Creates a new instance of vList Xtra linked to a file fully downloaded into the internet cache. You must first download the UR using preloadnetthing and wait for netDone to return true before trying to read from the cached URL file.

Save the vList instance returned by this command into a variable and pass it as an argument to read. This command simply establishes the path to the target file. If the target file does not yet exist, or is not yet fully downloaded, no error will occur unt read is issued.

You can read from a cached URL but you cannot write to it or delete it.

Example:

```
property inProgress,netID,url

on beginSprite me
inProgress = false
end

on mouseUp me
url = "http://www.macromedia.com/STORAGE.LST"
netID = preloadNetThing(url)
inProgress = true
end

on exitFrame me
if inProgress = false then exit
if netDone(netID) = true then
inProgress = false
if netError(netID) = 0 or (netError(netID) = "OK") then
netFile = new xtra("vlist",url)
alist = netFile.read()
end if
end if
end
```

vlistInstance.write(dataToStore,[encryption key]) - where vlistInstance is the instance previously created by the new method, dataToStore is a Lingo list or any other supported data type and encryption key is a linear list containing between 6 and 16 integers (Optional parameter). Returns 0 if successful, or a negative number in case of error. This error code can be misleadin and it is better in case of an error to look at the value returned by vList_error or vList_errorString. Do check the error status af a write. An unsuccessful read is usually due to a previous unsuccessful write.

Writes out a list, or any other supported data type, to the filepath the instance is linked to. Creates the file if it does not exist. I the file name supplied does not have a .LST extension, vList adds the extension when it creates the file. If the file already exis this method overwrites the existing file. Use fileExist to prevent accidental overwrite.

Note: This command only works with local files. It returns an error if used with a cached URL file.

Shockwave

Under Shockwave, if the size of the new data you are writing exceeds 128K, the Xtra shows a dialog that allows the user to cancel the operation. In this case this command returns a negative value you can check with vList_error and vList_errorString the user allows the operation no further disk space limit is imposed on data written during the session.

Examples:

```
fileLink = new xtra("vlist","C:\TEMP\STORE.LST")

alist = [1,2,3]

if fileLink.write (aList) <> 0 then

put vList_errorString() into field "status"

end if



-- with optional encryption

fileLink.write(alist,[54,22,240,98,5,6])
```

vlistInstance.read([encryption key]) - where vlistInstance is the instance previously created by the new method and encryptior key is a linear list containing between 6 and 16 integers (Optional parameter). Returns the contents of the file.

Reads in a list, or any other data type that has been stored, from the file the instance is linked to. If the list contents have been encryptedthe contents are decrypted with the provided key. If the list contents have been compressed, vList detects the compression automatically and decompresses the contents. There is no explicit decompress command.

If the vList instance is linked to a cached URL the file must be fully downloaded first using the Lingo command preloadNetThing call in order for the read to work.

Note: If you use this command to read from a file not created by vList it will return an error.

Examples:

```
fileLink = new xtra("vlist","C:\TEMP\STORE.LST")

thelist = fileLink.read()



-- to decrypt a list encrypted with key [54,22,240,98,5,6]

thelist = fileLink.read([54,22,240,98,5,6])
```

new(#vList) - Returns the newly-created vList member. The new member command is part of Lingo, and not technically a vL command. You can use Lingo's new command to create a new cast member of any type.

vList Xtra adds a new cast member type of #vList which stores Lingo lists. Use the new command to create vList members on the fly in Lingo, or select Insert -> Database -> vList from Director's authoring menu.

Example:

```
newListMem = new(#vList)

put newListMem

-- (member 1 of castLib 1)
```

Content Property of vList Member

vList members have many useful properties but the most important one is "content". A vList member's content property conta the data it stores. You set and read the content property of a vList member to access its stored data, which most likely will be a list but can be any type of supported data. Either dot syntax or "the propertyname of" syntax will access the content property o vList member. The syntax "put list into member vListMember" will also write a list to a vList member.

Remember to check the error status after writing content to a vList member. An unsuccessful read is usually due to a previous unsuccessful write.

When you set a variable to the content property of a vList member that contains compressed data, vList detects the compressio
decompresses the data on the fly, and returns the decompressed data. There is no explicit decompress command.

Examples:

```
listStorageMember = new(#vList)

put member(listStorageMember).content

-- < Void >

member(listStorageMember).content = ["a","b","c"]

put the content of member(listStorageMember)

-- ["a","b","c"]

put [1,2,3] into member(listStorageMember)

put member(listStorageMember).content

-- [1,2,3]



-- put member("vList member").content into aVariable

localList = member("vList member").content
```

ENCRYPTION

vList Xtra employs AES (Advanced Encryption Standard) encryption , the standard for secret-key encrypted transfer of unclassified information recently adopted by NIST, the US National Institute of Standards, as a replacement for DES. vList X encrypts list data with a block cipher, and a key length of 128 bits. (Psssst ... if your eyes are starting to glaze over, there is a good intro to cryptography that explains these concepts here.) This is the same key size used by the strongest level of Netscape SSL (Secure Socket Layer) encryption, the standard that protects credit card numbers and other private information in a brows In fact, 128-bit encryption is so strong that many countries, including the United States, place restrictions on the export of applications that use it.

The list of integers required as an argument in vList Xtra's encrypt methods is used for the key. The maximum integers allowe in the list is 16 (16 * 8 bits per integer = 128 bits). The minimum is 6. Since 16 are needed for a 128 bit key, vList fills in any are not passed, by itself. If more than 16 are passed, no error is returned but vList will only use the first 16. Since only 8 bits a allowed for each value, the maximum allowed value of any integer on the list is 255. If a larger value is passed the Xtra does a modulo 256 on the value to get a value it can use. For example 300 would be converted to 44.

Starting with version 1.8.6, vList includes a function (vList_encryptionStrength) that lets developers reduce the encryption lev
used by the Xtra to 64-bit keys, if necessary due to export restrictions. Again, please consult the encryption section for more information on this.

vList Xtra does not store the encryption key within an encrypted list file or list member, so it is very important to make a back
of the content of your encrypted members or list files. If you forget the encryption key, there is NO way to retrieve the content

AES encryption is very secure, but your list content can be recovered if someone guesses or uncovers your secret encryption k
It is essential that you take precautions to protect your key.

Most people will opt to store the key in the Lingo code of the movie file that needs to access the encrypted files or members. T
is the easiest thing to do, and there are some tips following to help you keep the key safe.

vList Xtra does not restrict you to any particular form of key handling. If you do not want to store the key inside your Director
movie, here are some examples of alternate schemes for encryption key-handling.

- The key is in a separate file, either local or remote, with the path to the key or an assumption about the relative path in the co

- The user enters the key each time. It is not stored.

- The key is on removeable media or a dongle that is inserted for use but stored elsewhere.

Steps to follow in order to protect your encryption key in Director movie files

When you protect your movies, Director strips the text of the Lingo code, and in the process also the code used to pass your encryption keys to vList Xtra methods. So the first precaution is to deliver your movies in protected form (*.DXR or *.CXR files) or, even better, in Shockwave compacted form (*.DCR or *.CCR files). Protected Lingo scripts are the only kind of information that lurkers and hackers have, for the most part, been unable to retrieve and study. All of the other Director memb
types are left unprotected and are fairly easy to access. This includes the content of pictures or 3D objects inside protected or even Shockwave compacted movies. This situation is the very reason why vList Xtra offers an encryption function, in order to
fully protect the content of the information you deliver inside a Director file.

vList Xtra uses a Lingo list of integer numbers to pass the key instead of strings. For example an encryption key like "passwor
should be passed to vList as:

```
member (listMem).encrypt ("this is the text to store", [112, 97, 115, 115, 119, 111, 114, 100])
```

A list is used for the key instead of a string because string variables are visible even in a protected movie file. If a string variab
was used for the key instead, someone could use a binary editor on your movie file, even protected by Director, and retrieve yo
keys, because strings are easily identifiable in a stream of binary data that otherwise is similar to a random succession of numbers. But you should go a step further and use lists of numbers that have no special meaning when translated to characters
This will make the life of eavesdroppers much more difficult. For example the previous key, even passed as a list of numbers i
stored in a movie file by Director as:

Å pÅ aÅ sÅ sÅ wÅ oÅ rÅ d

As you can see we can still clearly recognize the word "password", so it is better to use a sequence of numbers that has no meaning as a word:

```
-- this is better than using numbers that form in sequence a meaningful word or expression

encryptKey = [187, 200, 55, 99, 44, 21]

member (listMem).encrypt ("this is a text", encryptKey)
```

But even if you do so, Director still stores initialization code that create a recognizable pattern, with numbers separated by the character "Å", as we can see from the preceding binary translation. So we also advise you to generate your keys at runtime, wi Lingo code, instead of hardcoding the numbers, even if those numbers have no special meaning as character code:

```
-- this one is better

encryptKey = []

append encryptKey, 187

append encryptKey, encryptKey[1]+13

append encryptKey, encryptKey[2]-encryptKey[1]+42

...

member (listMem).encrypt ("this is a text", encryptKey)
```

Another measure of protection is to deactivate the Lingo tracing mode when creating your key:

```
encryptKey = generateMyKey(aNumber)

member (listMem).encrypt (aList, encryptKey)
```

The generateMyKey handler could be written as:

```
on generateMyKey aNumber

oldTrace = the trace

the trace = false
```

```
encryptKey = []

append encryptKey, aNumber

append encryptKey, encryptKey[1]+15

append encryptKey, encryptKey[2]-encryptKey[1]+100

...



the trace = oldTrace

return encryptKey

end
```

This way you can use the same code for the creation of the encryption and the decrytion key, and as long as you pass the same seed number, a Lingo handler will always returns the same result. You can create integer numbers greater than 255 and still us them because vList Xtra will apply a modulo operation that also will give always the same result:

1040 modulo 256 is equal to 528 modulo 256 which is equal to 16

But if you use a division operation in your key generating code, you must remember to cast the result to an integer, because Director will create a float number as a result of some code like:

```
append encryptKey, 100 / 3

-- a float number is stored in the encryptKey list. This is not allowed

-- so vList Xtra will return an error

append encryptKey, integer (100 / 3)

-- better, the result is truncated to the integer part and stored as 33
```

Also don't use global variables to store your key. If you do store a key list in a variable, be sure to use only a local variable.

Finally, although vList allows you to enter a minimum of six 8-bit values, this is for convenience only during development. Fo complete protection, we encourage you to make full use of all 128 bits of encryption, and pass all sixteen 8-bit values to the vl encryption commands.

In summary, for the best security:

- Convert your movie files containing encryption keys to protected or Shockwave format

- Do not use integers that form a word

- Do not store your key in a global

- Turn trace off before setting your key

- Generate your encryption key at runtime with Lingo code

- Pass all 16 encryption values

vlistInstance.write(dataToStore,encryption key) - Encrypts data and writes it to a vList file. There is no separate command to encrypt a file. You encrypt a file by adding an encryption key parameter to the write command.

read(vlistInstance.encryption key) - Decrypts data contained by the file and returns the decrypted data. There is no separate command to decrypt a file. You decrypt a file by adding an encryption key parameter to the read command.

member(vList member).encrypt(dataToStore,encryptionCodeList) - where dataToStore is a Lingo list or any other supported d type and encryptionCodeList is a linear list containing between 6 and 16 integers. Encrypts and then stores the specified data i the vList member.

Example:

alist = [#dog,#cat,#bird]

listMem = new(#vList)

member(listMem).encrypt(alist,[45,90,232,159,45,12,65])

member(vList member).decrypt(encryptionCodeList) - where encryptionCodeList is a linear list containing between 6 and 16 integers. Retrieves and decrypts the list contained in the vList member.

Example:

decryptedList = member(listMem).decrypt([45,90,232,159,45,12,65])

vList_encryptionStrength(encryptionBits) - where encryptionBits is 64 or 128. vList operates by default with 128-bit keys for encryption, but this can be reduced globally to 64, if necessary to comply with local export restrictions in some countries. This method was introduced in version 1.8.6, previous releases of vList shipped with 64-bit and 128-bit versions available as separa Xtras.

Example:

vList_encryptionStrength(64)

COMPRESSION

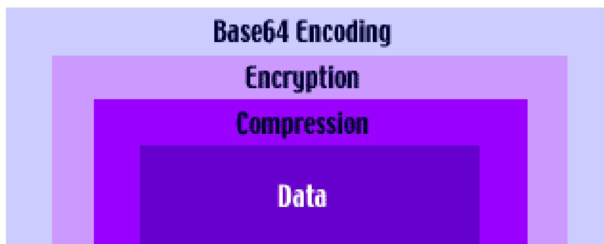
vList Xtra offers optional compression/decompression of data contained in vList members or files, using the ZIP algorithm. vl Xtra does not create standalone ZIP files. It creates files in vList format that contain compressed data. vList files containing compressed data can only be read by vList Xtra. vList Xtra cannot read standalone ZIP files.

You compress data by first turning compression on for an empty file or member vList container, then putting the data you wan compress into the container. Activating compression for an empty file or member causes subsequent data written to the contai to be compressed before it is stored.Setting the compression property to true or false via Lingo does not affect data already sto in the container.

You don't have to do anything special to decompress data in a vList container. When vList Xtra performs any read operation o file or member, it detects whether or not the data is compressed and automatically decompresses the data before returning the result of the read.

Combining compression with encryption and Base64 encoding

You can combine compression with encryption and Base64 encoding. The three are always applied in the same order, represented by the diagram below:



Compression looks for redundancy in the data while encryption eliminates it, attempting to produce a random data stream. Compressing encrypted data would result in a very low compression ratio, therefore compression is always applied to the unchanged data. Base64 encoding must always be the last conversion applied, since it serves as the "envelope" that allows the data to be transmitted unchanged over the internet.

When writing data, apply the three methods in this order:

1. Compress

2. Encrypt

3. Base64 Encode

When reading data,vList commands perform the reverse process automatically:

1. Base64 Decode

2. Decrypt

3. Decompress

The b64_decode function decodes a Base64 string, then decrypts the resulting data, if you pass a decryption key as one of the arguments. It then scans the decrypted data for a flag that indicates that it is compressed, and decompresses it if necessary. The read (file, key) and decrypt (member, key) functions also automatically detect compression in decrypted data and perform decompression before returning the data.

Activating compression for a member containing encrypted data returns an error message because the sequence for applying the two methods is out of order. The correct sequence is:

mem = new (#vList)

mem.compression = 1

mem.encrypt (someData, encryptionKeyList)

To compress member data that is already encrypted, do the following:

1. Decrypt and read the content into a variable using decrypt (member, key)

2. Put "" into the member to erase the current data

3. Activate compression for the member

4. Put the decrypted content back in to the member

The following handler illustrates these steps:

```
on compressEncryptedMember memName, encryptKey

data = member (memName).decrypt (encryptKey)

put empty into member(memName)

member(memName).compression = true

member(memName).encrypt (data, encryptKey)

end
```

Compressing bitmaps

For #bitmap members, applying compression to the stored properties member().media or member().picture results in about the same space savings for either one. If you store and compress the member's image property instead, you will get better space savings.

put member ("bitmap").image.duplicate() into member "vList"

vListInstance.compression(compressionFlag) - where vlistInstance is the instance previously created by the new method and compressionFlag is true to activate compression, false to deactivate compression. No return. Use the compressed function to determine the current status. Activates or deactivates compression for the file. Function affects subsequent writes to the file but does not change any data currently contained in the file.

Example:

```
xTraInstance = new xtra("vlist", "myfile")

-- activate compression

xTraInstance.compression (1)

xTraInstance.write (someData)

if xTraInstance.compressed () then

put "everything OK"

else

alert "data not compressed. problem"

end if

-- how much we gain on disk with compression

put xTraInstance.compressionRatio()

-- 60.5%
```

member(vList member).compression = compressionFlag - where compressionFlag is true to activate compression, false to deactivate compression. Activates or deactivates compression for the member. Function affects subsequent writes to the meml but does not change any data currently contained in the member.

MIME BASE64 ENCODING

Base64 encoding is a method of encoding data using only 6-bit characters for e-mail or internet transmission. MIME (Multipurpose Internet Mail Extensions) defines standards for transmitting internet messages and the various kinds of text and binary data they can include.

You can use Base64 encoding to transform a binary vList file into a string that Director's postNetText command can send to a CGI script on a remote web server. The receiving CGI script can then store the data by one of the following methods:

- Base64 decode the data and saves the resulting bytes into a binary file on the server. Since the data is a complete vList file before Base64 encoding, the receiving CGI script does not need any special logic to reconstruct the vList file.It must only dec the Base64 string and save the raw data to a binary file, which will then be a valid vList file.

- Save the Base64 string to a file without decoding it. Later on when the file is retrieved from the server, use vList's b64_deco function to decode it. This is less efficient because the stored Base64 files will be larger than the vList files they contain, so th will take longer to download when retrieved.

Sending Base64 strings via postNetText

Every character in the string returned from b64_encode must be transmitted unchanged by postNetText, or it will not decode i a valid vList file later on. The following options in postNetText must be set correctly to insure accurate transmission by postNetText:

Date sent as string, not property list: When you pass the data to be sent as a property list, postNetText URL-encodes some of t characters as though the data were coming from a form. Do not use the propertyList option to transmit a Base64 encoded strin because it may change some of the characters in the string.

ServerOS parameter must be set to "Win": The server OS parameter changes line ending characters in the data to the line endi used on the specified server platform. If you do not set this parameter it defaults to UNIX which will change all the line endin to just LF. You MUST set this parameter to "Win" regardless of what platform the receiving web server is on, otherwise the CR/LF (Win style) line endings in the Base64 string will be changed by postNetText and it will not decode correctly.

Trouble-shooting failed transmission

Using postNetText to transmit data to a CGI script successfully requires a good working understanding of network Lingo programming, the HTTP protocol, and CGI programming. You can further insure success by keeping the limitations of Direct postNetText in mind: do not try to use it in versions of Director or Shockwave prior to 8.5, or on the Mac in any environment other than Shockwave. If the data does not transmit and decode successfully on the server, here are some things to check:

Verify the data sent: Save the text string created by b64_encode using FileIO Xtra before transmitting it. Use any utility that c decode MIME Base64, such as Stuffit Expander, to verify that the resulting file is a valid vList file.

If the CGI is decoding the Base64 string, determine whether or not the CGI's Base64 decode function requires a MIME heade Some library functions for Base64 decoding accept only the Base64 data itself and will error out or produce no output data if t are passed a MIME header. Pass "" in b64_encode's filename parameter to get a Base64 string with no MIME header.

Allow the server time to decode and save the file before trying to retrieve it. A very large file may take some time to decode a save. If you try to retrieve it before the process has finished you may get only a partial file or a network error.

Verify the data received: Build debugging code into your CGI to help you verify that you are receiving the data intact:

1. Compare the environment variable "Content-length" to the length of the string you receive from the POST. They should be identical.

2. Compare the character values of each character received by the CGI to those sent (charToNum function in Lingo). This step will reveal any character substitution being done by postNetText. If this is happening, your postNetText options are not set correctly.

3. Use netTextResult: If you send debug output from your CGI, you can examine if from Lingo after netDone() returns true by using the netTextResult() function.

4. Save the Base64 string to a file before decoding: If your CGI is doing the Base64 decoding, add code to your CGI that saves the raw Base64 string to a file before decoding. Download the file to your local hard drive and use any utility that can decode MIME Base64, such as Stuffit Expander, to verify that the resulting file is a valid vList file. This step will help you narrow do whether the problem is with getting the data intact, or with decoding it.

b64_encode_compress(data,fileName,compressionFlag,|encryptionKey or optionsList|) - where:

data is a Lingo list or any other supported data type;

fileName is the string filename to use in MIME header, or empty string. The filename is used by MIME decoders which will s the decoded file using the specified filename.Limited to 24 characters including the file extension. If an empty string is passed no MIME header will be added to the returned data. If the filename is passed with no file extension, vList will add a .LST extension;

compressionFlag is a boolean (1 or 0) value indicates whether or not to compress the data before encoding;

encryption key or optionsList is a linear list containing between 6 and 16 integers or options property list. Optional parameter.

Returns a string containing a vList file encoded using MIME base64 encoding

Creates a vList file in memory (optionally compressed and / or encrypted) from the passed-in data, then MIME Base64 enco the vList file and returns it as a string. This command accepts 4 params at most. If you want to use encryption and set the binaryMode flag, use a property list to hold both values:

```
b64_encode_compress ("data to store", "myfile.lst", 1, [#encryptKey: [1,2,3,4,5,6,7,8], #binaryMode: 1] )
```

All of the following are valid syntax:

```
encryptionKey = [1,2,3,4,5,6,7,8,9,10,11,12]

b64_encode_compress (aLingoValue, filename, compressionFlag )

b64_encode_compress (aLingoValue, filename, compressionFlag, encryptionKey] )
```

b64_encode_compress (aLingoValue, filename, compressionFlag, [#binaryMode: 1])

b64_encode_compress (aLingoValue, filename, compressionFlag, [#encryptKey: encryptionKey, #binaryMode: 1])

b64_encode_compress (aLingoValue, filename, compressionFlag, [#encryptKey: encryptionKey])

vList's b64_encode function returns a string with a MIME header and Base64 Content-Transfer encoding if you pass a file nar
in parameter two. Here is what the encoded data looks like:

put b64_encode_compress([1,2,3],"myfile.lst",0)

-- "MIME-Version: 1.0

Content-Type: application/octet-stream; name="myfile.lst"

Content-Transfer-Encoding: base64

Content-Disposition: attachment; filename="myfile.lst"

AAAAAQAAAr463mixAAAAAwAAADgAAACMAAAABwABAAAAAAAAAQAAAAAAAAAAAAAAAAAXAAA

AGgAAAAAAAAATBMHB9EAAAAAAAAAAII7npKl1hHUv4gAUOSQMhoAAAAHAAAAAAAAAAEAAAAH

AAAAAwAAAAEAAAABAAAAAQAAAAEAAAACAAAAAQAAAAM=

"

If you pass an empty string for the filename parameter, the b64_encode function will return only the Base64 encoded data witl
no MIME header:

put b64_encode_compress([1,2,3],"", 0)

-- "AAAAAQAAAFU63mixAAAAAwAAAEgAAACMAAAABwABAAAAAAAAAQAAAAAAAAAAAAAAAAAXA

AGgAAAAAAAAATBAHB9EAAAAAAAAAAII7npKl1hHUv4gAUOSQMhoAAAAHAAAAAAAAAAEAAAAH

AAAAAwAAAAEAAAABAAAAAQAAAAEAAAACAAAAAQAAAAM="

Examples:

-- Compresses list "animals" because the compression flag parameter is 1, then

```
-- creates a vList file in memory out of the compressed list, then converts the file
-- into a MIME Base64 encoded string, which will be returned into variable aString.
-- When aString is later decoded, the decoder will name the resulting file
-- "animallist.lst". vList added the .lst extension to the file name.
--
animals = [#dog,#cat,#bird]
aString = b64_encode_compress (animals,"animallist", 1)


-- Converts variable data into a one-item list, encrypts the list, then converts the
-- list into a vList file in memory, then converts the file
-- into a MIME Base64 encoded string, which will be returned into variable aString.
-- When aString is later decoded, the decoder will name the resulting file
-- "imagedata.lst". The resulting will be an encrypted vList file.
--
data = member(15).media
aString = b64_encode_compress (data,"imagedata.lst", 0 ,[10, 20, 30, 40, 50, 60, 70])


-- Base64 encoding combined with postNetText
--
on mouseUp me
url = "http://www.domain.com/cgibin/filercgi"
listtosend = [1,2,3,4,5]
sendString = b64_encode_compress(listtosend,"", 0)
postNetText(url,sendString,"Win")
end
```

b64_encode(data,fileName,|encryptionKey or optionsList|) - where:

data is a Lingo list or any other supported data type;

fileName is the string filename to use in MIME header, or empty string. The filename is used by MIME decoders which will s
the decoded file using the specified filename.Limited to 24 characters including the file extension. If an empty string is passed
no MIME header will be added to the returned data. If the filename is passed with no file extension, vList will add a .LST
extension;

encryption key or optionsList is a linear list containing between 6 and 16 integers or options property list. Optional parameter.

Returns a string containing a vList file encoded using MIME base64 encoding.

Creates a vList file in memory (optionally encrypted) from the passed-in data, then MIME Base64 encodes the vList file and
returns it as a string. See b64_encode_compress for more information. This command accepts 3 params at most. If you want to
use encryption and set the binaryMode flag, use a property list to hold both values:

b64_encode ("data to store", "myfile.lst", [#encryptKey: [1,2,3,4,5,6,7,8], #binaryMode: 1])

All of the following are valid syntax:

encryptionKey = [1,2,3,4,5,6,7,8,9,10,11,12]

b64_encode (aLingoValue, filename)

b64_encode (aLingoValue, filename, encryptionKey])

b64_encode (aLingoValue, filename, [#binaryMode: 1])

b64_encode (aLingoValue, filename, [#encryptKey: encryptionKey, #binaryMode: 1])

b64_encode (aLingoValue, filename, [#encryptKey: encryptionKey])

Example:

put b64_encode([1,2,3],"myfile.lst")

-- "MIME-Version: 1.0

Content-Type: application/octet-stream; name="myfile.lst"

Content-Transfer-Encoding: base64

Content-Disposition: attachment; filename="myfile.lst"

AAAAAQAAAr463mixAAAAAwAAADgAAACMAAAABwABAAAAAAAAAQAAAAAAAAAAAAAAAAXAAA

AGgAAAAAAAAATBMHB9EAAAAAAAAAAII7npKl1hHUv4gAUOSQMhoAAAAHAAAAAAAAAAEAAAAH

AAAAAwAAAAEAAAABAAAAAQAAAAEAAAACAAAAAQAAAAM=

"

listData = b64_decode(MIMEencodedString,|encryptionKey or optionsList|) - where MIMEencodedString is a MIME Base64 string previously created by vList and encryption key is a linear list containing between 6 and 16 integers, or property list for multiple params (Optional parameter).

Decodes a MIME Base64 string created by vList, into a vList file in memory. Then decrypts the file if it is encrypted, extracts content property (the list or other data stored in the file) , decompresses the content if it is compressed,and returns it into the variable. This command accepts 2 params at most. If you want to decrypt and set the binaryMode flag, use a property list to ho both values:

b64_decode (encodedString, [#encryptKey: [1,2,3,4,5,6,7,8], #binaryMode: 1])

All of the following are valid syntax:

encryptionKey = [1,2,3,4,5,6,7,8,9,10,11,12]

b64_decode (encodedString)

b64_decode (encodedString, encryptionKey)

b64_decode (encodedString, [#binaryMode: 1])

b64_decode (encodedString, [#encryptKey: encryptionKey, #binaryMode: 1])

b64_decode (encodedString, [#encryptKey: encryptionKey])

Note: this function was designed to decode Base64 strings created by vList Xtra. Possibly a future version will add the ability decode MIME Base64 strings created by other sources.

Example:

thelist = [1,2,3]

encrypted_then_encoded_string = b64_encode(thelist,"",[10, 20, 30, 40, 50, 60, 70])

put encrypted_then_encoded_string

-- "AAAAAQAAAr463mixAAAAAwAAADgAAACMAAAABwABAAAAAAAAAQEAAAAAAAAAAAAAAAAAAXAA

AGgAAAAAAAAATB8HB9EAAAAAAAAAAIzGlRp1Seh1UEaQNVrsaydgmBX/TmuhLaXB2/xQ/M4z

5ngUgYAvp58qr0JBz0G4ArB51LitXv2MPaBsZOOxXAK=

"

get_it_back = b64_decode(encrypted_then_encoded_string,[10, 20, 30, 40, 50, 60, 70])

put get_it_back

-- [1,2,3]

READING AND STORING BINARY DATA

The following commands allow vList to read and create files of binary data. Previously vList was only able to read from files vList format. You can use these commands to do things like:

- Read in a media file, store it internally in a vlist member, then recreate the media file at a later date

- Copy files

- Read a local file in and transmit it using postnettext or ShockFiler Xtra

Null byte

Lingo requires a string to end with a null (0) byte character in order for operations like length(string) and char x of string to wo correctly. To accommodate Lingo, vList's readBinary function automatically adds a null character to the the end of the binary

data it is reading in. It does not add a null to a binary string it writes out with writeBinary. This behavior should work transparently both for binary files you plan to read and write using only vList and for binary Files you write for other applicati to read.

If you need behavior other than the default (writeBinary - no null, readBinary - add null), use the nullFlag optional parameter with readBinary and writeBinary to control whether or not the null bytes is appended to the binary string. Null bytes are not ar issue for the normal read or write commands.There is no nullFlag param for those commands.

IMPORTANT: In beta versions 1.6.5 and 1.6.7 the behavior for readBinary and writeBinary was reversed. WriteBinary wrote byte to the end of every file and readBinary did not add any byte to the string it read in. This behavior caused problems becaus some programs reading the binary files with the extra 0 character would not read the files. Also Director wants any string it rea in to be terminated with the 0 character. So it made sense to change the default behavior from:

1.6.5,1.6.7

writeBinary - added 0 char

readBinary - did not add any 0 char

to

1.6.9

writeBinary - does not add 0 char

readBinary - adds a 0 char to the string it reads in

If you are having crash problems reading in data you stored with 165 or 167 writeBinary or readBinary into vlist container, the easiest thing to do is to store the data again using 169 and the default null setting. Or you can use the null setting to turn the nu character on or off depending on how the data was previously stored.

vlistInstance.writeBinary(binaryDatastring,[nullFlag]) - where:

vlistInstance is the instance previously created by the new method;

binaryDatastring is the string containing binary data;

nullFlag is Optional. 1 to append a null character to the binary string written out to the file, 0 not to append null. Default is 0 i this param is not passed;

Returns 0 if successful, or a negative number in case of error. This error code can be misleading, and it is better in case of an error to look at the value returned by vList_error or vList_errorString. Do check the error status after a write. An unsuccessful read is usually due to a previous unsuccessful write.

Creates or overwrites a binary file at the path specified for the instance. Unlike the write method, this method does not create a vList file containing the data. It creates a binary file made out of the data in binaryDatastring. So the data in binaryDatastring must be sufficient to create a self-standing file of the type desired.

Note: an Xtra has to be intentionally written to manage binary data stored in a Lingo string so that Director doesn't truncate th data on a 0 byte. If you are reading data from any source other than readBinary or BinaryIO Xtra into a Lingo variable, and lat writing it out using writeBinary, please use lengthBinary(binaryString) first, to make sure that the length of the data is correct.

-- read in a JPEG file

vListInstance = new xtra("vlist", "image.jpg")

aString = vListInstance.readBinary ()

if not stringP(aString) then

put "the file was not read"

abort

end if

-- now write under another name

vListInstance = new xtra("vlist", "image2.jpg")

res = vListInstance.writeBinary (aString)

if res < 0 then

put "the file was not written"

abort

end if

readBinary(vlistInstance.[nullFlag]) - where:

vlistInstance is the instance previously created by the new method;

nullFlag is Optional. 1 to append a null character to the binary string read from the file, 0 not to append null. Default is 1 if thi
param is not passed;

Returns the contents of the file. Reads data from a binary file into a Lingo string variable. If the vList instance is linked to a cached URL the file must be fully downloaded first using the Lingo command preloadNetThing call in order for the read to w

Example 1

```
-- Display a dialog and read in file

vListInstance = new xtra("vlist", "image.jpg",-1)

binaryData = vListInstance.readBinary ()

storageMem = new(#vlist)

-- store data in vlist member

storageMem.content = binaryData
```

Example 2:

```
property inProgress,netID,url

on beginSprite me

inProgress = false

end

on mouseUp me

url = "http://www.macromedia.com/picture.gif"

netID = preloadNetThing(url)

inProgress = true

end


on exitFrame me

if inProgress = false then exit
```

```
if netDone(netID) = true then

inProgress = false

if netError(netID) = 0 or (netError(netID) = "OK") then

netFile = new xtra("vlist",url)

binData = netFile.readBinary()

localFile = new xtra("vlist","localpict.gif")

localFile.writeBinary(binData)

end if

end if

end
```

lengthBinary(binaryDatastring) - where binaryDatastring is a Lingo string variable containing binary data. Returns an integer length of the binary data in the variable. Use this instead of the Lingo length() function to return the length of a binary string variable. Most of the time this value will be one more than the Lingo length function for the same variable, but sometimes it w not match it at all.

```
y = new xtra("vlist","file.pdf")

bin = y.readBinary()

put lengthBinary(bin)

-- 8810
```

vListInstance.binaryMode(boolean)

member("vlist").binaryMode = boolean - where vlistInstance is the instance previously created by the new method and boolea true to turn on binary storage for future writes. No return.

vList, by default, performs cross-mapping on characters above numToChar(127) in strings stored inside its vList files and members. This enables strings containing only text to display the correct accented characters on Mac and PC. The default

cross-mapping behavior is not desireable for strings containing binary data because it will corrupt the data. Turn off the defaul cross-mapping behavior by turning the binaryMode option on for a vList file or member before putting binary data into it.

Note: You do not need to use binaryMode on an instance that you use to access a non-vList file. Data read using readBinary o written using writeBinary is always handled as binary. This option is only relevant for vList containers.

```
-- an external vList file

vListInstance = new xtra("vlist", "image.gif")

aBinaryString = vListInstance.readBinary()

vListInstance = new xtra("vlist", "storage.lst")

-- we are going to store binary data in a vlist file

vListInstance.binaryMode (true)

vListInstance.write (aBinaryString)



-- a vList member

mb = new (#vList)

mb.name = "aBinaryMember"

mb.binaryMode = true

mb.content = aBinaryString
```

The binaryMode option works for read as well. You can turn cross-mapping on and off to read the same data string with high-ASCII characters cross-mapped for the platform and without.

```
on writeMac

astring = numToChar(128)

vListInstance = new xtra("vlist", "storage.lst")

vListInstance.write (astring)

end
```

on readonPC_WithMapping

vListInstance = new xtra("vlist", "storage.lst")

data = vListInstance.read()

alert(string(charToNum(data))) -- 196

end

on readonPC_WithoutMapping

vListInstance = new xtra("vlist", "storage.lst")

vListInstance.binaryMode(true)

data = vListInstance.read()

alert(string(charToNum(data))) -- 128

end

If the original data was written with binaryMode on, you can still turn binaryMode off to read it in. This may be useful in the c
where the binary file contains fields of text data at known offsets. In that case you can do two reads, one as binary to preserve
file, and one with binary turned off into a second variable. You can then extract the cross-mapped text from the second variabl

Check the vList container's binaryContent property to determine whether or not it was written with binaryMode turned on.

FILE UTILITY OPERATIONS

The following commands require an instance of vList linked to a file, which is created with the new command.

vlistInstance = new xtra("vlist","datafile.lst")

vlistInstance.fileExist([encryption key]) - where vlistInstance is the instance previously created by the new method and encryption key is a linear list containing between 6 and 16 integers (Optional parameter). Returns 1 if a file at the path specifie
previously in the new command, exists and it is a vList file, 2 if it exists but it is not a vList file, or 0 if it doesn't exist. If the optional encryption key parameter is passed, the function will only return 1 if a vList file exists at the path and the file is

encrypted with the specified key.

Examples:

```
-- File "C:\TEMP\FOLDER\DATA.LST" does not exist at all

myFile = new xtra("vlist", "C:\TEMP\FOLDER\DATA.LST")

put myFile.fileExist()

-- 0


-- A vList file exists at "Work:Project:foo.lst" with an encryption key

-- of [213,222,89,23,43,12,45]


myFile = new xtra("vlist", "Work:Project:foo.lst",[213,222,89,23,43,12,45] )

put myFile.fileExist([213,222,89,23,43,12,45])

-- 1


-- Sequence of commands to check for readable encrypted files

--

fil = new xtra("vlist", "vlistFile.lst")

if fil.fileExist() = 1 then

-- we have a valid vList file here

if fil.encrypted() then

if fil.fileExist(myEncryptKey) then

-- file is encrypted and we have a valid key

else

-- encrypted, but we have forgotten the key

end if
```

else

-- the file exists and is not encrypted

end if

else

-- no vList file, we can safely create one by that name

end if

vlistInstance.deleteFile() - where vlistInstance is the instance previously created by the new method. Returns 0 if successful, o negative number in case of error (this error code can be misleading, and it is better in case of error to look at the value returned by vList_error or vList_errorString). Deletes the file at the path specified previously in the new command. This command will delete any file by the specified name, not just vList files. Note: This command only works on local files. You cannot use this command successfully if the vList instance is linked to a URL file in the internet cache.

Example:

```
-- Delete the vList file at "C:\TEMP\FOLDER\DATA.LST"

--

myFile = new xtra("vlist", "C:\TEMP\FOLDER\DATA.LST")

myFile.deleteFile()
```

bytesWritten = vlistInstance.fileSpace( ) - where vlistInstance is the instance previously created by the new method. Returns a Integer amount of data written to new files during this session, in bytes.

This function is useful in Shockwave to monitor the amount of data you are writing so that you will know when you are gettin near the limit (128K) that will trigger a security dialog that asks the user for permission for further writes. The function reports data written to vList files during one session, a session being the period of time the user keeps the browser page open that contains the Shockwave movie that loaded vList Xtra. Going to amother movie that loads into the same page, continues the session. Going to another HTML page containing another movie, from the original movie, ends the session.

The filespace function, tells you how many bytes that count toward the limit of 128K (131072 bytes), you have already writter other files during the session. The filespace function is tied to the current file instance. It is telling you, "This is the number of bytes counting toward the limit that are already used, if you write to the current file.".

The value can be positive or negative. A negative value is a credit. To calculate the maximum number of bytes you can write t
the current file without triggering the Shockwave dialog:

inst = new xtra("vlist","foo.lst")

val = inst.filespace()

limit = 131072

safeWriteBytes = limit - val

So if val is -30000, which is a 30000 credit, 131072 - (-30000) results in a total allowed write of 161072. (131072 + 30000 cre

The filespace value returned for an open file instance does not change for the instance while you have it open, unless you dele
the file. Checking it before and after a write will return the same value. If you open another file, check filespace again before
writing, to get an updated value for the current file that takes into account what you wrote to the previous file.

The filespace value is cumulative. If you create file1 and write 100K to it, then create file2 in the same Shockwave session an
attempt to write 29K or more to it, you will trigger a user permission dialog.

If you open and write to an existing file you have a credit for the entire size of the file, even if the file is larger than the 128K
limit. The reason for this exception is that if the file is already on disk, then writing data less than or the same size as the data
already there cannot cause disk space problems for the user.

Any new bytes you write to an existing file during the current session are subtracted from the credit when you close the file an
open another file. If FileA exists and is 100 bytes long, you have 100 bytes of credit as long as you keep writing to FileA. If yo
write 50 bytes to FileA, then open FileB, the 50 new bytes are subtracted from the credit and you now have 50 bytes of credit
apply to future writes.

Note that vList does not check to see whether or not a file is a vList file when a "new" command is issued. If you use new to
create an instance of vList linked to a non-vList file, then do inst.fileSpace(), it will show a credit for the non-vList file.

Action	Filespace value
Create FileA	0
Write 50 bytes to FileA	
Create FileB	50
You have written 50 bytes to A toward the limit	
Open existing FileA size 100 bytes	-100
Write 50 bytes to A	
Create FileB	-50
Opening existing FileA gave a 100 byte credit. The 50 bytes of new data written to A was subtracted from the credit, leaving a 50 byte credit.	
Open existing FileA size 100 bytes	-100

Write 150 bytes to A	
Create FileB	50
The 150 new bytes written to A was offset by the 100 bytes credit, leaving a net of 50 new bytes written this session.	
Open existing FileA size 100 bytes	-100
Delete FileA	0
Deleting a file erases its credit.	
Open existing FileA size 100 bytes	-100
Write 50 bytes to A	
Delete FileA	50
If FileA had not been deleted, its credit would have been applied and offset the 50 new bytes written for a net of -50, but deleting the file erased the credit and the entire 50 bytes count toward the limit.	

Example:

```
inst = new xtra("vlist", "aFile")

alist = [1,2,3]

temp = new(#vlist)

temp.content = alist

totalSize = inst.fileSpace() + member(temp).sizeOnDisk

if (totalSize < 128*1024) then

-- no shockwave warning dialog

inst.write (aList)

else

-- shockwave warning dialog will be shown

-- if we write to the file, so do something else

end if
```

VLIST MEMBER OPERATIONS

The following operations borrow familiar syntax usually associated with fields and strings to alter the contents of vList memb
Adding data with these methods to a sorted list stored inside a vList member returns the list to an unsorted state.

Note: It is important to use full member syntax with these commands.

Correct

```
memberRef = new(#vList)

put list after member(memberRef)
```

Incorrect

```
memberRef = new(#vList)

put list after memberRef
```

put data into member(vList member) - Puts a list or any data value into the specified vList member. (Lingo only)

Example:

```
put [1,2,3] into member("listStorage")

put member("listStorage").content

-- [1,2,3]

put 2551 into member("listStorage")

put member("listStorage").content

-- 2551
```

put aList after member(vList member) - Appends the data to the vList member's current content. (Lingo only).

If the current content of the vList member is a list, the data to append must also be a list of the same type otherwise vList will return "handler not defined". If the two lists are the same type, the items contained by the passed-in list are appended to the existing list. If the current content of the vList member is some other data type, vList will convert its content to a linear list consisting of the original data item followed by the one to append.

Examples:

```
put member("listStorage").content

-- [1,2,3]

put [4,5,6] after member("listStorage")

put member("listStorage").content

-- [1,2,3,4,5,6]



put member("listStorage").content

-- [#dog:1,#horse:2]

put [#camel:8] after member("listStorage")

put member("listStorage").content

-- [#dog:1,#horse:2,#camel:8]



put member("listStorage").content

-- 85

put 47 after member("listStorage")

put member("listStorage").content

-- [ 85,47 ]



put member("bitmap").media into member("listStorage")

put member("listStorage").content

-- (media fad8890)
```

put 2 after member("listStorage")

put member("listStorage").content

-- [(media fad8890), 2]

put aList before member(vList member) - Prepends the data to the vList member's current content. (Lingo only)

If the current content of the vList member is a list, the data to prepend must also be a list of the same type otherwise vList will return "handler not defined". If the two lists are the same type, the items contained by the passed-in list are put before the item the existing list. If the current content of the vList member is some other data type, vList will convert its content to a linear lis consisting of the passed-in data followed by the original data item.

Examples:

put member("listStorage").content

-- [1,2,3]

put [4,5,6] before member("listStorage")

put member("listStorage").content

-- [4,5,6,1,2,3]

put member("listStorage").content

-- [#dog:1,#horse:2]

put [#camel:8] before member("listStorage")

put member("listStorage").content

-- [#camel:8,#dog:1,#horse:2]

put member("listStorage").content

-- 85

put 47 before member("listStorage")

put member("listStorage").content

-- [47,85]

member(vList member).importFile(| filePathString |) - where filePathString is either a full path to the vList file to import or ju
file name. If just a file name is passed, vList looks for the file in the current movie's directory (Optional argument). Returns 0
successful, or a negative number in case of error (this error code can be misleading, and it is better in case of error to look at th
value returned by vList_error or vList_errorString).

Imports the list contained in a vList file into a vList member. If filePathString is not passed, puts up a file picker dialog. This command is not available in Shockwave.

Example:

-- puts up a file picker and imports the chosen vList file

importFile(member("listStorage"))

-- imports vList file "scores.lst" in the movie's directory

importFile(member("listStorage"),"scores.lst")

-- imports the vList file at the specified path

importFile(member("listStorage"),"Macintosh HD:games:scores.lst")

mediaContents = vList_readFromMedia(vListMediaProperty[encryptionKey]) - where vListMediaProperty is the the media
property of a vList member either stored in a variable or obtained from member(vlist).media and encryption key is a linear list
containing between 6 and 16 integers (Optional parameter). Returns the data contained in the vList media.

Provides a shortcut for retrieving data from vList members that have been stored inside other vList members or files by storing
(vList member).media. Also provides a workaround for a problem with setting the media property of a vList member in a repe
loop.

Example:

-- one vList member stores the media of several others

testList = list (member("vList1").media, member("vlist2").media)

member("storage").content = testlist

-- later, we want to retrieve the data

-- Long way to extract data

scratch = new(#vlist)

scratch.media = member("storage").content[1]

dataWeWant = scratch.content

erase scratch

-- Short way

dataWeWant = vList_readFromMedia(member("storage").content[1])

OKflag = vList_checkMedia(vListMediaProperty,[encryptionKey]) - where vListMediaProperty is the the media property of a vList member either stored in a variable or obtained from member(vlist).media and encryption key is a linear list containing between 6 and 16 integers (Optional parameter). Returns 1 if the media is intact, 0 if it is corrupt. Utility function helpful in diagnosing a problem with setting the media property of a vList member in a repeat loop.

Example:

-- one vList member stores the media of several others

testList = list (member("vList1").media, member("vlist2").media)

member("storage").content = testlist

-- later, we want to retrieve the data

scratch = new(#vlist)

storedVlistMedia = member("storage").content[1]

if vList_checkMedia(storedVlistMedia) then

scratch.media = storedVlistMedia

end if

LIST OPERATIONS

The following vList commands operate on Lingo lists.

concatenate(list, list, ..., list) - Returns a list containing the list elements of the second, third, and so on ... list(s) appended to th end of the first list Appends the elements of the second (or more) list argument(s) to the end of the first list. All the lists must b the same type - property or linear.

Concatenates the lists passed in, in place, which destroys the original lists. If there is an error, this command displays an error dialog in authoring mode, which is consistent with Director's handling of errors on list operations while in authoring. Howeve unlike Director, vList does not display the error dialog in projectors or in Shockwave mode.

Example:

a = [1,2,3]

b = [4,5,6]

c = concatenate(a,b)

put c

-- [1,2,3,4,5,6]

put a

-- [ ]

put b

-- [ ]

insertAt(linearList,positionNumber,anyValue) - where positionNumber is the integer index position to insert the item and anyValue is the data to insert into the list. Returns 0 if successful, or a negative number in case of error (this error code can be misleading, and it is better in case of error to look at the value returned by vList_error or vList_errorString).

Also if there is an error, this command displays an error dialog in authoring mode, which is consistent with Director's handlin errors on list operations while in authoring. However unlike Director, vList does not display the error dialog in projectors or i Shockwave mode.

Requires Director 8 or above. Inserts a new item into a linear list at the specified position, moving the items after the designat position up one index position. Specifying an index position that is the count of the list plus 1, appends the item to the list.

Example:

theList = [1,2,3]

insertAt(theList,1,"dog")

put theList

-- ["dog",1,2,3]

insertAt(theList,5,"cat")

put theList

-- ["dog",1,2,3,"cat"]

insertAt(theList,3,"canary")

put theList

-- ["dog",1,"canary",2,3,"cat"]

insertPropAt(propertyList,positionNumber,anyProp,anyValue) - where positionNumber is the integer index position to insert item, anyProp is the property to insert and anyValue is the value to pair with inserted property. Returns 0 if successful, or a negative number in case of error (this error code can be misleading, and it is better in case of error to look at the value returne by vList_error or vList_errorString).

Also if there is an error, this command displays an error dialog in authoring mode, which is consistent with Director's handlin errors on list operations while in authoring. However unlike Director, vList does not display the error dialog in projectors or i

Shockwave mode.

Inserts a new property/value pair into a property list at the specified index position, moving the items after the designated position up one index position. Specifying an index position that is the count of the list plus 1, appends the new property/value pair to the list. Using this command on a sorted list turns the sort flag off.

Examples:

theList = [#leafColor:"green",#barkColor:"white",#barkTexture:"rough"] insertPropAt(theList,1,#height,53)

put theList

-- [#height:53,#leafColor:"green",#barkColor:"white",#barkTexture:"rough"] insertPropAt(theList,5,#flowering,0)

put theList

-- [#height:53,#leafColor:"green",#barkColor:"white",#barkTexture:"rough",#flowering:0] insertPropAt(theList,3,#attractive,

put theList

-- [#height:53,#leafColor:"green",#attractive:1,#barkColor:"white",#barkTexture:"rough",#flowering:0]

setPropAt(propertyList,positionNumber,anyProp) - where positionNumber is the integer index position to insert the item and anyProp is the new property. Returns 0 if successful, or a negative number in case of error (this error code can be misleading, it is better in case of error to look at the value returned by vList_error or vList_errorString).

Also if there is an error, this command displays an error dialog in authoring mode, which is consistent with Director's handling errors on list operations while in authoring. However unlike Director, vList does not display the error dialog in projectors or in Shockwave mode.

Changes the property at the specified index position of a property list to the new property passed in. Leaves the value associate with the property unchanged. Using this command on a sorted list turns the sort flag off.

Example:

theList = [#leafColor:"green",#barkColor:"white"]

setPropAt(theList,1,#flowerColor)

put theList

-- [#flowerColor:"green",#barkColor:"white"]

changeProp(propertyList,oldProp,newProp,deepMode) - where oldProp is the property to change, newProp is the new value fo the property and deepMode is a boolean, true to do the replace at all levels of a property list with sublists, false to operate on j the top level. Returns the number of properties found and replaced, 0 if no matches for oldProp were found, or a negative num in case of error (this error code can be misleading, and it is better in case of error to look at the value returned by vList_error o vList_errorString).

Also if there is an error, this command displays an error dialog in authoring mode, which is consistent with Director's handling errors on list operations while in authoring. However unlike Director, vList does not display the error dialog in projectors or in Shockwave mode.

Changes all of the properties in a property list matching oldProp to newProp in the property passed in. Leaves the value associated with the property unchanged. Operates on the top level of a multi-level list if deepMode is false. If deepMode is tru operates on all levels. Using this command on a sorted list turns the sort flag off.

Example:

theList = [#leafColor:"green",#barkColor:"white"]

numChanged = changeProp (theList,#barkColor,#flowerColor,0)

put theList

-- [#leafColor:"green",#flowerColor:"white"]

put numChanged

-- 1

If the properties we want to change are not on the first list level, we have to set the deepMode to true:

aPropList = [[#a: 1], [#b: 2, #a: #a]]

numChanged = changeProp (aPropList, #a, #c, false)

put aPropList

-- [[#a: 1], [#b: 2, #a: #a]]

put numChanged

-- 0

numChanged = changeProp (aPropList, #a, #c, true)

put aPropList

-- [[#c: 1], [#b: 2, #c: #a]]

put numChanged

-- 2

appendProp(propertyList,anyProp,anyValue) - where anyProp is the property of the property/value pair to insert and anyValue the value of the property/value pair to insert. Returns 0 if successful, or a negative number in case of error (this error code can misleading, and it is better in case of error to look at the value returned by vList_error or vList_errorString).

Also if there is an error, this command displays an error dialog in authoring mode, which is consistent with Director's handling errors on list operations while in authoring. However unlike Director, vList does not display the error dialog in projectors or in Shockwave mode.

Appends a new property/value pair to the end of a property list.

Example:

theList = [#a:"dog",#b:"cat"]

appendProp(theList,#y,"mouse")

put theList

-- [#a:"dog",#b:"cat",#y:"mouse"]

unsort(list) - No return. Turns off the sorted flag for the list. Does not actually reorder the list contents.

Example:

aList = ["jam","bread","toast"]

sort(aList)

put aList

-- ["bread","jam","toast"]

put sortP(aList)

-- 1

unsort(aList)

put sortP(aList)

-- 0

put aList

-- ["bread","jam","toast"]

duplicateSort(list) - Returns a list that is a copy of the one passed in. Copies a list, preserving its sorted state. duplicateSort do not actually sort the contents of the list.

Director's duplicate command also makes a copy of a list, but in Director 8, it resets the sorted flag to false for the new list and any nested lists within it. duplicateSort is a fix for that problem in Director 8. It is not necessary for Director 7 or Director 8.5 where the duplicate command works correctly.

Example:

aList = ["jam","bread","toast"]

sort(aList)

put aList

-- ["bread","jam","toast"]

put sortP(aList)

-- 1

newList = duplicateSort(aList)

put sortP(newList)

-- 1

-- Must do this in Director 8 to see problem

anotherList = duplicate(aList)

put sortP(anotherList)

-- 0

CREATING POPULATED LISTS

The following methods create lists with a fixed number of identical items initialized with filler data. These methods are especially useful if you know beforehand the size of a Lingo list you want to use. If you create a list with dummy values at initialization time, the entire list will be stored in contiguous memory if there is enough available, which keeps Director's memory unfragmented.

You could accomplish the same thing by using Lingo repeat loop to build the dummy list, but the repeat loop method is very slow. The newList commands are much faster than using a Lingo repeat loop, so there is no trade off of a long delay at startup exchange for efficient memory use.

newList(numberOfItems,fillValue) - where numberOfItems is an integer, the number of list items to create and fill with fillVa and fillValue is an integer or symbol, the value for list items. Returns a linear list with the specified number of items, each fill with an identical value.

Example:

sectionScores = newList(10,0)

put sectionScores

-- [0,0,0,0,0,0,0,0,0,0]

ticTacToeBoard = newList(9,#empty)

put ticTacToeBoard

-- [#empty,#empty,#empty,#empty,#empty,#empty,#empty,#empty,#empty]

newPropList(numberOfItems,fillProperty,fillValue) - where numberOfItems is an integer, the number of list items to create an fill with the property/value pairs, fillProperty is an integer or symbol, the value to use for the property of the property/value pa and fillValue is an integer or symbol, the value to use for the value of the property/value pair. Returns a property list with the specified number of items, each filled with an identical property/value.

Example:

recordSet = newPropList(3,#untitled,0)

put recordSet

-- [#untitled: 0, #untitled: 0, #untitled: 0]

LIST/DATATYPE INFORMATION

The following functions return information about Lingo lists and other supported data types.

sortP(list) - Returns 1 if the list has its sorted flag set, 0 if not. Returns the state of the list's internal sorted flag. If the flag is fa
the actual list contents may or may not be ordered.

Example:

-- Don't do a time-expensive sort operation unless it is necessary

if sortP(myList) = false then sort(myList)

isSame(variable,variable) - Returns 1 if the two lists, or other supported data types, point to the same memory location. When you pass data from one handler to another or copy data from one variable to another sometimes Director makes a new copy of (stored by value) and sometimes it passes a pointer to the memory location of the existing data (stored by reference).This function tells you if two Lingo variables containing data types that are passed by reference are actually pointers to the same memory location, or two separate copies.

Some data types passed by reference are image(), rect, list, member.media, member.image, string (D8.5 and up)

Some data types passed by value are integer, float, symbol

This function is only useful for comparing data passed by reference. If it is passed data types stored by value it compares their contents and works like the = operator. For instance if you use it on two integer variables it simply returns whether their value are equal.

A = 5

B = A

put isSame(A,B)

-- 1

Example:

A = list(1,2,4)

B = A

put isSame(A,B)

-- 1

C = duplicate(A)

put isSame(A,C)

-- 0

numRef(variable) - Returns the number of references to the memory location storing the variable, or 0 if the data type is not stored by reference. This function tells you how many references to a variable currently exist. The information can be useful fo memory management because Director cannot release a variable for garbage collection if there are still other references to it.T function uses the same convention for counting references that Director itself does. The data itself counts as one reference. Th first set of a variable to the data makes the reference count 2.

aList = [1,2,3]

put numRef (aList)

-- 2

bList = aList

put numRef (bList)

-- 3

aList = 0

put numRef (bList)

-- 2

put bList

-- [1, 2, 3]

Integer and property Lingo variables are not count referenced, so the method returns 0 for these kinds of variables:

put numRef (1)

-- 0

put numRef (#Lingo)

-- 0

Example:

on startMovie

var = new(script "parent")

alert "Before anotherHandler: " & (numRef(var))

anotherHandler(var)

alert "After anotherHandler, its local ref discarded: " & (numRef(var))

end

on anotherHandler var

local = var

alert "In anotherHandler: " & (numRef(local))

end

-- Parent script "parent"

property foo

on new me

return me

end

float32P(float) - Returns 1 if the value is a 32-bit float, 0 if it is a 64-bit float, or some other data type. Returns the type of stor
required for the particular floating point number. Function only exists in Director 8.5 and above, which introduced the more compact 32-bit float.

In Director 8.5 floats with up to 8 digits before the decimal and 0 after the decimal only require 32 bits of storage. vList Xtra stores 32-bit floats in their native format in vList members and files created under Director 8.5.

Example (running under D85):

put float32P(123.0)

-- 1

put float32P(123.123)

-- 0

float32(number) - Returns a 32-bit float, if the value passed in is a float or an integer. Otherwise returns the passed in value unchanged. Converts a float into a value that Director can store in 32 bits. Function only exists in Director 8.5 and above, whi
introduced the more compact 32-bit float. The passed-in value will be rounded off if it contains too many digits to be stored in
bits.

Lingo operations on 32-bit values are up to 50% faster than those on 64-bit values. However, Director automatically converts floats to 64-bit, so it is difficult to maintain the smaller storage size if a variable containing a 32-bit float has to be put through
series of Lingo operations.

Example (running under D85):

put float32P(8.123456789)

-- 0

z = float32(8.123456789)

put z

-- 8.1235

put float32P(z)

-- 1

vList_size(list) - Returns the integer size in bytes of the list, or any other supported data type. Returns the projected size of a Lingo list after it has been stored in a vList container and loaded back in to memory. This figure could differ from the list's current size before storage, if the list contains data types such as #script, #xtra, and #window that vList cannot save. Each data reference that cannot be saved, is represented by vList as an 8-byte void value.

Examples:

put vList_size(myList)

-- 88484

put vList_size("dog")

-- 60

vList_sizeOnDisk(list) - Returns the Integer file size of the list, or any other supported data type, if it was written out to a file. Returns the space in bytes that a Lingo list currently in memory would take up if it were written out to a vList file.

Examples:

put vList_sizeOnDisk(myList)

-- 88500

put vList_sizeOnDisk(member(32).picture)

-- 716

EXPORTING DATA TO SHOCKFILER XTRA

ShockFiler Xtra saves data from a Shockwave movie or a projector to a standalone file on an FTP server. vList Xtra includes support for exporting vList data through ShockFiler Xtra to a standalone vList file on an FTP server.

propList = vList_to_sf(data,|compressionFlag|,|encryptionKey|)

propList = vList_to_sf(data,|optionsList|) - where:

data is a Lingo list or other data type;

compressionFlag is true to compress the data. Optional parameter;

encryptionKey is a linear list of 8-16 integers (encryption key) Optional parameter;

optionsList is a property list containing all of, or at minimum, one of, the following:

[#compression: 1, #binaryMode: 1, #encryptKey: [1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8]]

Returns a property list containing data packaged in a format that can be used by ShockFiler Xtra.

This function packages vList data into a format that ShockFiler Xtra can export to a standalone vList file on an FTP server. Th packaged data is returned in a Lingo property list in the format:

[#data: binary vList data, #length: size of data in bytes]

vlist_to_sf can only accept a total of three parameters. If you just want to pass an encryption key, you can use the original sim syntax:

propList = vList_to_sf (data, 1, [1,2,3,4,5,6,7,8])

put propList

-- [#data: "", #length: 2233388]

-- binary data will display as an empty string due to control chars

If you need to pass a binaryMode flag as well, use the newer alternate options list syntax to pass it:

propList = vList_to_sf (data, [#compression: true, #encryptKey: [1,2,3,4,5,6,7,8] , #binaryMode: true])

All of the following lines are valid syntax:

encryptionKey = [1,2,3,4,5,6,7,8,9,10,11,12]

aList = vList_to_sf (aLingoValue)

aList = vList_to_sf (aLingoValue, false)

aList = vList_to_sf(aLingoValue, true, encryptionKey])

aList = vList_to_sf (aLingoValue, [#compression: true, #binaryMode: true])

aList = vList_to_sf (aLingoValue, [#compression: true, #encryptKey: encryptionKey, #binaryMode: true])

See the ShockFiler Xtra doc for more details.

Example:

vlistData = vList_to_sf ([#name: "Andy", #score: 50] , 1, [1,2,3,4,5,6,7,8])

-- ShockFiler command

sf_Send ("ftp.macromedia.com", "incoming","guest", "mypassword",vlistData, "", "vfile.lst", #bina, [], 0, 0,"")

MISCELLANEOUS

fpReset( ) - Windows command that resets the floating point register after it has been left in a bad state by a printer driver. Thi
situation caused a vList crash when reading and writing files, and causes the same type of crash in other xtras and Director itse
The crash dialog identifies this type of crash as an "exception 10H".

vList now resets the FP register internally so this command is not necessary to protect vList operations. But it can be useful to
head off trouble in other xtras or Director. For instance it has been reported to fix a Flash printing problem from a Flash sprite
Director.

The following links provide more information on the problem:

http://www.aresforact.com/support/1001.htm

http://support.microsoft.com/default.aspx?scid=kb;EN-US;q183522

setFileName(memberRef,fileName) - where memberRef is a reference to a member and fileName is a string, name of file to li
to in Shockwave, or full path in projector. This is a workaround method to substitute for Lingo's member.fileName property, which no longer works in Shockwave on Mac.Previously, doing the following would link the member to "doc.jpg" in the DSWMEDIA folder:

member("pict").fileName = "doc.jpg"

Now you can do this instead:

setFileName(member("pict") , "doc.jpg")

This command also works in a projector. It looks in the current movie's folder for the file when running in authoring or in a projector.

vList_version( ) - Returns a string containing the version number of vList Xtra.

Example:

put vList_version()

-- "1.8.6"

ERROR REPORTING

Many vList methods return data, so they cannot also return a status. Do not rely on the return value of any call other than "register" for error returns. Instead, use the vList_error function to get the status of the last vList call.

vList_error( ) - Returns the integer number of the last vList error. Returns 0 if the previous vList call was successful, or a negative number if there was a problem. Use vList_errorString(errorNumber) to get a description of the error.

Note: This call resets the internal error code, so be sure to store the result of this command in a variable in case you want to pa
it later to vList_errorString().

Example:

```
fileIOInstance = new(xtra "fileio")

chosenFile = displayOpen(fileIOInstance)

fileLink = new xtra("vlist", chosenFile)

storedList = read(fileLink)

if vList_error() = -2147211504 then

alert "Cannot retrieve data. The chosen file exists but it is not a vList file."

end if
```

vList_errorString(|errorNumber|) - Returns the string error description. Returns a description of the last error, if no argument i
passed. Returns the definition for a specific negative error number if one is passed in. The error description returned is in the
format:

ErrorOrigin_ErrorDescription (ErrorNumber)

kMoaErr_NoErr: (0)

kLstErr_Not_A_PropList (-2147211489)

The first 4 characters of the error origin tell you whether the error came from MOA, Director's Xtras API, ("kMoa") or from v
itself ("kLst"). The rest of the string after the first underscore and before the space describes the error condition. The number
between the parentheses is the error number.

Example:

```
put vList_errorString()

-- "kMoaMmErr_ArgOutOfRange (-2147229496)"
```

In the example above, vList returned the error from the last vList operation because no argument was passed.

put vList_errorString()

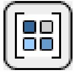
-- "kLstErr_Not_A_PropList (-2147211489)"

In the example above, vList returned the error from the last vList operation because no argument was passed. The error came from vList.

put vList_errorString (-2147221484)

-- "kMoaErr_InternalError (-2147221484)"

In the example above, a particular error number was passed in and vList returned the error description for that error.



VLIST XTRA HELP: MEMBER AND FILE PROPERTIES DOCUMENTATION

vList members have the following properties that you can access using normal Lingo syntax for getting properties.

propertyValue = member("vlist member").propertyName

propertyValue = the propertyName of member("vlist member")

Example:

variable = member("vlist").platform

You can get most of the same information about vList files by using slightly different syntax.

propertyValue = vlist_instance.propertyName()

Example:

fileLink = new xtra("vlist","data.lst")

variable = fileLink.platform()

content - (Member property only) The data stored within the vList member. Members that have been created but have not yet had a list written to them have an initial content value of <Void> if their content type has been set to Empty or Unique Value in the properties dialog box, or [ ] or [:] if their content type has been set to Linear or Property.

Example:

put member("vlist member").content

-- [1,2,3]

dirVersion - The Director version the member or file was created under if no list has yet been written to it. Otherwise the Director version running when the content contained by the member or file was last written to it..

Examples:

put member("vlist member").dirVersion

-- "7.0.2"

put vlist_instance.dirVersion()

-- "8.5"

platform - The platform ("Macintosh" or "Windows") the member or file was created under if no data has yet been written to it. Otherwise the platform the movie was running on when the data contained by the member or file was written to it.

Example:

put member("vlist member").platform

-- "Macintosh"

put vlist_instance.platform()

-- "Windows"

count - The number of list items in the list contained by the vList member or file. If the data contained by the member or file is not a list, vList returns 1 for the property.

Example:

```
member("vlist member").content = ["a","b"]

put member("vlist member").count

-- 2



fileLink = new xtra("vlist","storage.lst")

fileLink.write("Here is a string to store")

put fileLink.count()

-- 1
```

size - Number of bytes of memory that would be taken up by the data contained by the vList member or file if it was loaded from the member or file into a variable. The amount of memory required by a list depends on the data types stored.

Example:

```
put member("vlist member").size

-- 24



fileLink = new xtra("vlist","storage.lst")

fileLink.write("Here is a string to store")
```

put fileLink.size()

-- 74

sizeOnDisk - Number of bytes of disk space that would be taken up by the data contained by the vList member if it was written out to a vList file.The disk size required by a stored list depends on the data types stored.

Example:

put member("vlist member").sizeOnDisk

-- 28

fileLink = new xtra("vlist","storage.lst")

fileLink.write("Here is a string to store")

put fileLink.sizeOnDisk()

-- 156

contentType - The type of data contained by the vList member or file. Returns the same values as the Lingo ilk function does for built-in date types. Lingo's ilk property correctly returns "#instance" for vList Xtra instances in memory, so this property is necessary to look inside a vList member or file and get information about what kind of data is stored inside.

Examples:

newMem = new(#vList)

put member(newMem).contentType

```
-- #void

member(newMem).content = member(5).media

put member(newMem).contentType

-- #media

member(newMem).content = [#a:1,#b:2]

put member(newMem).contentType

-- #propList



fileLink = new xtra("vlist","storage.lst")

fileLink.write(88)

put fileLink.contentType()

-- #integer
```

sorted - Whether or not the sort flag is on for the list contained by the vList member or file. Does not report the actual ordering of the contents. If the vList member or file does not contain a list, vList returns 0 for this property.

Examples:

```
put member("vlist member").sorted

-- 0



put instance.sorted()

-- 1
```

member ("vlist member").sort - (Member property only) Setting this property to true sorts the list contents and sets the sorted flag to true. Setting this property to false, turns off the flag but does not reorder the list contents. If the vList member does not contain a list setting this property does nothing.

Note: You cannot sort an encrypted list member by setting this property to true. Instead, first extract the list using decrypt, then use the built-in sort function on the list in memory.

Example:

member("vlist member").content = ["fish","dog"]

put member("vlist member").sorted

-- 0

member("vlist member").sort = true

put member("vlist member").sorted

-- 1

put member("vlist member").content

-- ["dog","fish"]

member("vlist member").sort = false

put member("vlist member").content

-- ["dog","fish"]

put member("vlist member").sorted

-- 0

encrypted - Whether or not the data contained by the vList member or file is encrypted.

Examples:

member("vlist member").content = ["fish","dog"]

put member("vlist member").encrypted

-- 0

listToStore = ["One phrase string"]

member("vlist member").encrypt(listToStore, [41, 82, 79, 255, 35, 23])

put member("vlist member").encrypted

-- 1

put instance.encrypted()

-- 1

fileName - (File property only) The full filepath to the file linked to the vList instance. The file itself may not yet exist if no write has yet occurred. In Shockwave, returns just the filename, not the full path.

Note: If you have not yet saved your movie in authoring there will be no moviePath to build the full file path from if you specified only a file name when creating the file link. In this case the function returns only the file name.

Example:

instance = new xtra("vlist","data.lst")

-- Since no path was specified the file will be created in the same

-- directory as the movie

put the moviepath

-- "Macintosh HD:Project:"

put instance.fileName()

-- "Macintosh HD:Project:data.lst"

compressed - Whether or not the data contained by the vList member or file is compressed.

Examples:

vlistMem = new(#vlist)

member(vlistMem).content = ["fish","dog"]

put member(vlistMem).compressed

-- 0

vlistMem = new(#vlist)

member(vlistMem).compression = true

member(vlistMem).content = ["fish","dog"]

put member(vlistMem).compressed

-- 1

afile = new xtra("vlist","datafile.lst")

afile.compression (true)

write(afile, ["fish","dog"])

put afile.compressed()

-- 1

compressionratio - The percentage of disk space saved by compression.

Examples:

```
vlistMem = new(#vlist)

member(vlistMem).compression = true

member(vlistMem).content =
member("bitmap").media

put member(vlistMem).compressionratio

-- 54.237
```

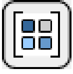
```
afile = new xtra("vlist","datafile.lst")

afile.compression (true)

afile.write( member("bitmap").media )

put afile.compressionratio()

-- 54.237
```

binaryContent - Whether or not the string data inside a vlist file or member will be crossmapped when it is read on the opposite platform. This property tells you whether binaryMode was turned on when the data was written to the container.

Examples:

```
-- file

inst = new xtra("vlist","%foo.lst")

inst.binaryMode(true)

inst.write(numToChar(255))

alert(string(inst.binaryContent())) -- 1
```

```
-- member

mem = new(#vlist)

mem.binaryMode = true

mem.content = numToChar(255)

alert(string(mem.binaryContent)) -- 1
```



VLIST XTRA HELP: SHOCKWAVE

vList Xtra can be used in Shockwave: the distribution package provides packaged Xtras that are downloaded automatically to the user's machine, and installed on demand. Please consult Adobe's web site for a complete overview of the Xtras automated download mechanism: read the Shockwave Xtras downloading overview technote. The basic steps required to make vList Xtra available for download are outlined below.

To create a Shockwave movie that will auto-download the Xtra to the user's hard drive you must do the following, in this order:

1. Upload the packaged Xtra files to your web server

2. Modify the entry for vList Xtra in file xtrainfo.txt to point to the packaged Xtra files on your server

3. Do Modify -> Movie -> Xtras, select vList, and check the "Download if Needed" option

Once you have completed steps 1 and 2, you can create other Shockwave movies by doing only step 3.

PACKAGED FILES

Your vList Xtra archive contains a subfolder called "Shockwave". There are four files inside it:

vList.w32 - Win 32 package

vList.ppc - Mac Classic package

vList.carb - Mac Carbon package

vList.xpku - Mac Universal Binary package

All packages contain the vList Xtra for that platform. Depending on the user's platform, a package autodownloaded to the user's hard drive will install the correct vList Xtra for the user's platform into their Shockwave support folder.

If, for some reason, you choose not to make your Shockwave movies autodownload the package files, you can have the user install the right Xtra for their platform into the Shockwave support folder manually.

Upload all package files to the same directory on your web server. Use a "binary" or "raw", not "text" transfer. If the packages are uploaded to two different directories, autodownloading will not work. Do not rename the package files.

If you are going to distribute vList with Shockwave movies, we recommend that you use your own web server to do so. Packages are available at Tabuleiro's download services, but we reserve the right to refuse access, without notice, to any referring URL that generates excessive traffic.

The vList Xtra packages included with the download have been signed and packaged by Tabuleiro, and will present the following security message to users of your Shockwave movies when they are installed for the first time:

You may choose to repackage vList Xtra and sign it with your own Verisign certificate. You might want to do this if you want your own company name to appear in the auto-download dialog box the user sees when an auto-download is initiated. Adobe is the best source of

information on applying for a Verisign certificate and packaging files.

XTRAINFO.TXT

The text file xtrainfo.txt resides in your Director authoring directory. It contains information about Xtras such as file version names for an Xtra on both platforms and the URL for the packages. The information contained in xtrainfo.txt is saved with each movie you create and used by projectors and Shockwave.

You must create an entry for vList Xtra in your xtrainfo.txt file that specifies the URL on your server for the package files. The last part of the path will always be "vList". That specifies the filename of the packages within the directory, without the file extension. Do not include a file extension at the end of the path.

```
[#namePPC:"vList", #nameW32:"vList.x32",
#package:"http://www.domain.com/folder/vList"]
```

Make sure that the line above does not contain any return character after you paste it into your xtrainfo.txt file. Open your text editor wide and make sure the line does not wrap. If the opening and closing brackets are not on the same line, Director will not be able to create a valid list from the entry and the "Download if needed" button will be dimmed for vList in Director.

If you edit xtrainfo.txt while Director is open you should quit and restart Director to read in the changed information in xtrainfo.

EDITING THE MOVIE'S XTRAS LIST

Open the Director movie that you want to save as Shockwave. Choose Modify -> Movie -> Xtras and add vList. Select vList from the list and check the "Download if needed" option. Director will initiate an internet connection and look for the packages at the URL you specified in xtrainfo.txt.

If Director finds the packages, it will transfer information about the package contents for both platforms such as file names and version numbers and embed the information into your Director movie. An informational dialog box will appear that tells you that the packages for both platforms are "downloading". The packages themselves are not downloading, just information about them that the Shockwave movie will need later to compare the version of the Xtra the user possibly already has to the version currently on the server in order to determine if autodownloading is necessary. The Director movie needs information about both platforms because it may find itself running on either platform once it is on the web.

Once "downloading" of the packages has finished, save the movie, then publish as Shockwave. The finished Shockwave movie can reside at any URL. It does not have to be in the same directory or even on the same server as the packaged Xtras.

If a connection cannot be opened, or the packages cannot be found at the specified location, Director will uncheck the "Download as needed" option automatically. You must have a successful connection for the box to remain checked. A Shockwave made out of a Director movie with the "Download as needed" button unchecked will not autodownload vList Xtra.

RUNTIME DIFFERENCES UNDER SHOCKWAVE

Some vList commands that access local files are disabled or work differently under Shockwave for security reasons. The following is an overview of the restrictions. Click on the link to each command for more specifics.

- new: Local files can only be written to and read from a Shockwave support folder or some predetermined System paths, without explicit permission from the user. It is not possible to specify any file path. However, you can use the dialog option in the new command to prompt the user for a path when a read or write is executed. The path is used internally by vList to write or read the file, but it is not available to Lingo.

The following table shows the sublist of system paths available for read and write in Shockwave:

Special folder name

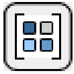
	Can use read/write	Can use readBinary /writeBinary
	The deleteFile command will delete only vList files	The deleteFile command will delete any file
Mac		
kDesktopFolderType	X	X
kSystemDesktopFolderType	X	X
kTrashFolderType	X	
kPreferencesFolderType	X	
kSystemPreferencesFolderType	X	
kTemporaryFolderType	X	
kDocumentsFolderType	X	
kApplicationSupportFolderType	X	
kPictureDocumentsFolderType	X	X
kMovieDocumentsFolderType	X	X
kMusicDocumentsFolderType	X	X
kPublicFolderType	X	X
kCurrentUserFolderType	X	
kSharedUserDataFolderType	X	
Windows		
CSIDL_DESKTOP	X	X
CSIDL_PERSONAL	X	X
CSIDL_BITBUCKET	X	

CSIDL_MYMUSIC	X	X
CSIDL_MYVIDEO	X	X
CSIDL_DESKTOPDIRECTORY	X	X
CSIDL_COMMON_DESKTOPDIRECTORY	X	X
CSIDL_APPDATA	X	
CSIDL_LOCAL_APPDATA	X	
CSIDL_COMMON_APPDATA	X	
CSIDL_COMMON_DOCUMENTS	X	
CSIDL_COMMON_MUSIC	X	X
CSIDL_COMMON_PICTURES	X	X
CSIDL_COMMON_VIDEO	X	X

- write , writeBinary: A permission dialog displays after 128K of new data has been written during a Shockwave session. If the user grants permission, any amount of additional data can be written. Overwriting existing files does not count toward the 128K limit unless the amount of new data written exceeds the amount of data originally in the file. Use the function fileSpace to monitor the limit. See the doc for fileSpace for a more detailed description of how the limit is calculated.

- fileInstance.fileName(): Returns just the file name, not path, in Shockwave

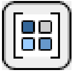
- importFile: Disabled in Shockwave



VLIST XTRA HELP: JAVASCRIPT

XTRAS AND JAVASCRIPT

Director MX 2004 added JavaScript as an alternative scripting language, and this is also available in Director 11. The syntax in the methods doc for vList Xtra works for both Lingo and JavaScript. Most of the methods in vList take or return Director lists or symbols as arguments. Lists and symbols are not native data types for JavaScript, but Director provides ways for JavaScript to work with these data types. For more info, see the tech note on JavaScript and Xtras.

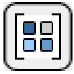


VLIST XTRA HELP: USING MEDIA OF MEMBER WITH VLIST

vList containers (file or member) can store Lingo variables that contain the media of a member. This can be very useful for example in a situation where you plan to update a Director project at runtime using the internet or a CD-ROM. You can do so by preparing your movie with placeholder members of the same type as those you plan to distribute in the future. At runtime you retrieve the new member media from a vList storage file and replace the movie's member media with the new media. The movie acts as a content shell so that only the changed media must be retrieved, which is more efficient.

vList members are normal Director members, so you can also store their media in a Lingo variable or store it in another vList container. In fact you can even store the media of a vList member inside itself, but you should be advised that what you store in that case is the previous state of that member:

```
memList = new (#vlist)

put 1 into member memList

memList.content = memList.media -- storing member().media into itself

memList.media = memList.content -- restoring to the previous state

put memList.content

-- 1
```



VLIST XTRA HELP: HOW DIRECTOR HANDLES LIST SORTING

All lists have an internal flag set to true or false that indicates to Director whether the list is currently in a sorted state or not. When Director searchs in a list (getProp, findPos, findPosNear), it first checks to see whether the sorted flag is true. If the list is sorted, then Director searches beginning at the middle of the list. If the value there is greater than the one to look for, then Director goes back and starts looking at the value located in the middle of the first middle part of the list, etc. until the target value is found and the position returned. In other words Director uses a standard binary search to find an item in a sorted list, which is usually more efficient than searching straight from beginning to end. If the list has its sorted flag set to false, Director's search begins with the first element, then the second, etc. until the value is found. This method is usually much slower, unless by chance, the target item is at the beginning of the list.

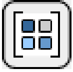
When you do:

sort (list)

Director does two things: the list is ordered (alphabetically) on its elements and the list's sorted flag is set to true.

When you store a sorted list as text with a string() conversion and then retrieve it back in memory with value(), the list is still in order, but the sorted flag is set to false. So searching inside the list is slow again, as slow as searching on an unordered list. If you want to have fast searchs, you are forced to do a sort(aList) again, which will take approximately half the time needed to sort an unordered list. What the vList Xtra does is to KEEP that sorted state on disk. When the list is read back into memory, vList lets Director know that the list is still in a sorted state WITHOUT the need to do a time-consuming sort(). So you have the ability to do a fast search of your list immediately after reading it into memory.

Note: There is one situation where sorting a list can actually slow things down. If the list is a property list, and the properties are symbols, Director temporarily converts the properties to strings to order them aphabetically. The same process happens when you search for a target symbol. Both the target symbol and the list's symbols are converted to strings and a string compare is done until a match is found.



VLIST XTRA HELP: HOW DIRECTOR STORES DATA TYPES

All data types are not created equal when it comes to memory storage. Some are stored by value, which means that a separate copy of the data is made each time it is used in the list, copied to a variable, or passed as an argument to a handler. Integers, floats and symbols are examples of data types that are stored by value. Other data types are stored by reference, which means that a pointer to the item's memory location is stored, instead of a new separate copy of the data.The data types member.media and member.image, image(), strings, rects and points are stored by reference.

If the item stored by reference appears in a list multiple times, multiple pointers to it are stored rather than multiple separate copies of it. You can see this by typing the following in the message window:

```
aList = [ ]

append aList, member(1).media

append aList, member(1).media

append aList, member(1).media

put aList

-- [(media eefe4e0), (media eefe4e0), (media eefe4e0)]
```

0EEFE4E0 is the address in memory where member(1)'s media is stored.

STRINGS

Strings are stored by value in Director 7 and 8. In Director 8.5 they are stored by reference. You can use vList's isSame function to determine if two variables are pointing to the same memory location or not.

```
A = "some text"

B = A
```

In Director 7 and 8, string B is a separate identical copy of A:

```
put isSame(A, B)

-- 0
```

In Director 8.5 both B and A are pointers to the same memory location:

```
put isSame (A, B)

-- 1
```

In Director 8.5 and later, if you go on to modify string B, Director then makes a separate copy of the original and stores it in B:

```
B = B & " more text"

put isSame(A, B)

-- 0
```

At this point A and B occupy two separate memory locations and the reference count for A, which still points to the original string, is decremented by 1.

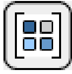
Making copies of large strings each time they were passed to handlers or copied to variables really slowed down some operations in previous versions of Director. This optimization of string storage in D8.5 is a welcome enhancement, that was designed to be transparent to Lingo coders.

In Director 11, strings are stored as Unicode (UTF8) data. vList will attempt to transparently convert strings to/from UTF8 and the older MacRoman/Latin1 character mappings, used until Director MX2004. However, some characters might not be translatable in double byte languages when going from Director 11 to older versions, and vice-versa. Also, strings that are part of the media of members will not be inspected or translated. These members need to be upgraded using the upgrade movies mechanism that exists in Director 11.

FLOATS

Prior to Director 8.5, Director reserved 64-bits (8 bytes) for each floating point variable. In Director 8.5 there is a more economical 32-bit (4 byte) floating point data type, which can store floating point numbers with up to 8 digits before the decimal and nothing (0) after the decimal. The storage size required for a float is determined transparently by Director and not normally apparent to the Lingo programmer. However, vList can distinguish between the two float types under Director 8.5 and stores the smaller 32-bit float in its native format both in vList members and on disk.

vList's float32P function returns the storage requirement of a float under Director 8.5.



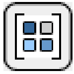
VLIST XTRA HELP: EXPORT OF AES ENCRYPTION

Tabuleiro is a Brazilian company and not subjected to restrictions in the sale and export of products capable of strong encryption. However, many countries including the United States restrict the export of products that contain strong encryption. The US export restrictions were relaxed quite a bit in fall 2000, but current US regulations still restrict sale of products that include 128-bit AES encryption, to customers residing in pre-approved countries only.

If you create a project that makes use of the AES encryption features of vList, your country may have regulations that apply to the export of your product. Some countries, including the United States, have more relaxed export rules for 64-bit or shorter keys, so a customer planning to export a product that does not require a 128-bit key, may prefer to use weaker encryption.

Previous releases of vList distributed by updateStage, Inc were available in two different versions, one using 64-bit keys and the other using 128-bit ones, in order to comply with US export laws that affected updateStage, Inc. Version 1.8.6 and later however includes a new method, vList_encryptionStrenght(64), that can be used by the developer to restrict the cryptographic capabilities of vList at runtime. If you live in a country that limits the export of products using strong encryption, you may want to use this function to effectively reduce encryption strenght in vList, thus making your final product that uses vList compliant with export regulations that permit weaker keys.

Export legislation is constantly being revised and is different for each country, so if you have any legal questions regarding your particular situation please consult your lawyer.



VLIST XTRA HELP: LIMITATIONS

- #date storage: Sometimes the seconds part of a date will be rounded up in a date retrieved from a vList file. Unfortunately the rounding error is inside MOA, so it can't be fixed. If you are going to need precise seconds stored inside a #date data type the best thing to do is store it in two parts in a list, with the seconds part obtained via Lingo before storage:

aDateList = [#date: dateObject, #seconds: dateObject.seconds]

- You cannot use a relative file name of 2 characters or less before the file extension with the read command. It will return a "File not found" error. Workaround: pass a full path. This will error:

inst = new(xtra "vlist","aa.lst")

data = inst.read()

This will work:

inst = new(xtra "vlist",the moviepath & "aa.lst")

data = inst.read()

- If you use the new 2004 syntax to call a handler in an MIAW that returns a list, the thing returned is a "list object" rather than a Lingo list. vList cannot work with list objects. For instance, an MIAW called "miaw" contains this handler in a movie script:

on getList

return [1,2,3]

end

The stage does this:

w = window().new("windowTitle")

w.filename = _movie.path & "miaw"

-- new syntax

```
listObject = w.movie.mGetList()

-- old syntax

tell w to normalList = mGetList()

put listObject

put normalList
```

And you get:

```
-- <Object list 3 6a34b90>

-- [1,2,3]
```

List objects were introduced in 2004 so that the Javascript engine could work with the list datatype.

- Storing sound sprites in a vList member or file may cause type 2 errors when you quit Director 8 in authoring, in a projector and may crashes your Browser in Shockwave 8 on the Mac. This is a Director problem that is fixed in Director/Shockwave 8.5, The workaround for Director 8 is to set variables referencing sound objects to 0 before quitting.

```
alist = list( sound(1) )

member("vList mem").content = alist

newList = member("vList mem").content <---- May cause type 2 error in D8

-- workaround to use before exit

newList = 0
```

- vList's character cross-mapping for stored strings relies on Director's fontmap.txt file. FONTMAP.TXT contains several mapping errors for high-ASCII characters (characters in the range numToChar(128) and above). Since many developers have workaround code in place that would break if these characters were mapped differently, vList Xtra uses FONTMAP.TXT's mappings for all characters. Notice that when running in Director 11 vList will store strings in UTF8 format, and will convert to/from UTF8 and Latin/MacRoman character mappings if necessary, if you attempt to use media from one version of Director in another.

- While it is possible to store the .image property of a member directly in a vList file or member, this actually stores a pointer to the image data. The problem is that if you go to another movie, this pointer will be useless. If you actually want to have an image object that can be used in any movie, you have to create a duplicate of the image data and work with that, like so:

```
newImage = duplicate(member("bitmap").image)
```

The newImage object is similar to one created using the Lingo command:

```
newImage = image(100, 100, 32) -- a 100 by 100 pixels with 32-bit colors
```

In this case what vList will store in a file or a member is the complete pixmap information needed to recreate that image object at runtime. Note that at the moment vList stores the independent image objects without any compression. So in the example above, vList will need at least 40.000 bytes to store that information. Use member().media to store your images if size is an issue. Director uses compression on bitmap members in order to reduce the size of movies on disk.

- Director gives you the option of decreasing the size of bitmap members even further, in movies converted to Shockwave, by applying JPEG image compression to them. vList cannot currently apply JPEG compression to bitmap members stored using media of member inside vList. Therefore a Shockwave movie comprised mostly of internal JPEG members will have a smaller file size if the members are not saved inside vList containers.

When you import a JPEG compressed file into a movie, Director creates a member with TWO images. The first one is the internal representation, that uses a lossless compression format on disk, and the second one is a copy of the JPEG compressed (lossy) original image. The lossless copy is necessary in case you want to edit the image.

When you save the file as a Shockwave movie (.DCR), Director strips the internal image and retains only the smaller JPEG one. When you save the file as a protected movie (.DXR), Director strips the smaller JPEG image and retains only the larger lossless compressed internal image. When vList stores member("bitmap").media for a member that uses JPEG compression, vList, like the DXR file, stores the larger lossless compressed internal image. This was a design decision of necessity that may impact Shockwave files that make heavy use of internal JPEG images. Solutions will be investigated for a future version, for instance, the addition of optional JPEG compression that could be selected for a stored member().image property. Meanwhile, one advantage of storing images in this format is that the larger internal image takes much less time to decompress at runtime than a JPEG-compressed image.

- postNetText: Director's postNetText command is limited to 64K of data on all platforms in versions prior to Director 8.5. In Director 8.5, in authoring and projector, on the Mac Classic platform only, postNetText corrupts strings longer than 40K (40960) by overwriting the data starting at byte 40961 with data from the beginning of the string. This is not a problem in Shockwave on the Mac, which relies on the browser to perform the post. It is not a problem in any environment, authoring, projector, or Shockwave, on Windows.

- If you use "put data after member 'vlist member'" syntax to add list items to a vList member, under certain rare circumstances a saveMovie will not be successful until the vList member contains at least 2 items. The problem exists only in Director 7. It is fixed in Director 8 and above.

Example:

-- member("vlist) starts out empty

repeat with x = 1 to 3

put #picture after member("vlist")

-- workaround code

if x >= 2 then

saveMovie

end if

end repeat

- Mac only. Under some undetermined circumstances, setting the media of one vList member to the media of another within a repeat loop will corrupt the media property so that the copy fails.

Example:

repeat with x = 1 to 3

member("vList" & x).media = member("vList" & (x + 100)).media

end repeat

Workaround is to use vList_readFromMedia to obtain the data and avoid the media property in this situation.

```
repeat with x = 1 to 3

data = vList_readFromMedia(member("vList" & (x + 100)).media)

member("vList" & x).content = data

end repeat
```

Although in the situation above you could always use the content property instead of the media property, it is possible to store the media of vList members in other vList members. In that case, when trying to restore the vList members, there is ONLY the media property and no content until a new member is created. vList_readFromMedia can help in the following situation:

```
alist = member("vList container for other vList

embers").content

put alist

-- [ <media 83838>, <media 83838>, <media 83838> ]

-- old method for getting vList data back out

repeat with x = 1 to 3

mem = new (#vList)

mem.media = alist[x]

dataYouWant = mem.content

-- do something with data, then erase dummy member

erase member mem

end repeat
```

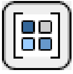
Another workaround from Gregory Lardon is to force a pause or add something that gives more time to Director. For example the following is problematic and leads to apparent media corruption:

```
repeat with i = 1 to n

maList.add(member("x").media)

end repeat
```

But this works if we add an intermediate step:

```
repeat with i = 1 to n

vMedia = member("x").media

maList.add(vMedia)

end repeat
```

Gregory also says that adding a put "" statement inside the loop was a solution to the problem. This problem is not present in Director under Mac OS X or Windows.

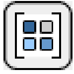


VLIST XTRA HELP: HOW TO ORDER & REGISTER

The unregistered version of vList Xtra is fully-functional. Free functions in the Xtra can be used for nonprofit, educational and commercial purposes, with no limitation. A registered version of the Xtra unlocks Basic or Full functionality, and may be purchased online at xtras.tabuleiro.com, using a secure server. At our web site you can also consult our purchase policy, purchase instructions, payment, delivery and security methods.

If you decide to buy the Xtra you don't need to download a new copy of the software. After your order is processed you will receive an e-mail with a serial number to register the software you've already installed on your machine, unlocking Basic or Full functionality.

To register the Xtra you should use the vList_Register() function, usually called at the startup of your movie, or before a vList Xtra function is used. More information about specific syntax can be found at the Methods Documentation page. Please keep your serial number archived for future reference.



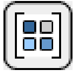
VLIST XTRA HELP: LICENSING & AVAILABILITY

vList Xtra is a commercial product. Current price and updated information can be found at xtras.tabuleiro.com. If your product provides printed documentation and package we ask you to kindly include the following copyright information:

vList Xtra(tm) (c) Tabuleiro Prod. Ltda 2008

All Rights Reserved

No royalty-fees are required for a distribution of the Xtra with your product.



VLIST XTRA HELP: TECHNICAL SUPPORT

Please use the Your Account section available at our web site xtras.tabuleiro.com to submit your questions. The site also contains Technotes and other resources that can help you identify and solve the most common problems quickly.