

OLE xtra Version 1

www.xtramania.com

All trademarked names mentioned in this document and product are used for editorial purposes only, with no intention of infringing upon the trademarks.

OLE xtra	1
About OLE xtra	1
About ActiveCompanionSet	1
What is the Difference	1
'ActiveCompanionSet' Xtras License Agreement	1
OLE xtra Programmer's Guide	5
Inserting new OLE cast member	5
Media editor	5
Scripting operations	7
Creating new OLE cast member	8
Inserting OLE Object	9
By User Choice	9
From File	9
New Empty Object of Specified Type	9
Editing OLE object	10
Editing OLE object with VbScriptXtra	11
Controlling OLE object appearance	12
Debugging and Errors Handling	13
Lingo Errors	13
Programming Errors	13
Simple Debugging Mode	13
Advanced Debugging Mode	13
OLE xtra Programmer's Reference	15
Asset-level	16
Error handling support	16
<i>Succeeded</i>	16
<i>Failed</i>	16
<i>LastErrorCode</i>	17
<i>LastError</i>	17
Debugging Support	18
<i>DebugMode</i>	18
Xtra Specific Properties	19
<i>Version</i>	19
<i>CLSID</i>	19
<i>ProgId</i>	19
<i>OLEState</i>	20
<i>StoredImageType</i>	20
<i>DrawWithOLE</i>	21
<i>OLEMediaChanged</i>	21
Xtra Specific Methods	22
<i>Open()</i>	22
<i>GetObject()</i>	22

<i>Save(symImageType)</i>	23
<i>Close(bSave)</i>	23
<i>LoadOLEObject()</i>	24
<i>UnloadOLEObject()</i>	24
<i>InsertOLE()</i>	25

OLE xtra

About OLE xtra

OLE xtra extends the Macromedia Director's Lingo functionality with capability to handle OLE embedded objects.

OLE xtra is available for Macromedia Director (v7 and later) under Windows 95/98/ME/NT/2000/XP.

OLE xtra is not available for Shockwave.

Note: All trademarked names mentioned in this document and product are used for editorial purposes only, with no intention of infringing upon the trademarks.

About ActiveCompanionSet

OLE xtra is shipped within ActiveCompanionSet. It is a bundle of xtras that provide COM, OLE and ActiveX support for Macromedia Director. The set currently includes VbScriptXtra, OLE xtra and ObjectBrowserXtra. Further versions of the ActiveCompanionSet will include ActiveX visual controls support.

What is the Difference

Macromedia Director ships with its own OLE xtra. ActiveCompanionSet provides even better support for OLE embedded objects and scripting. Below is the list of key features of the OLE xtra from ActiveCompanionSet.

- Stores OLE object data together with bitmap or metafile view.
- Draws object either with native OLE handler of the parent application (if any) or uses stored object's view.
- Supports scripting of OLE operations: creating and editing OLE objects at run-time with the native parent application (if any).
- Provides even further scripting support by means of VbScriptXtra (if embedded object supports scripting). For example with Microsoft Word embedded document it is possible to open embedded object with Word, edit it with Lingo scripting and then save it back to the Director's cast member.

'ActiveCompanionSet' Xtras License Agreement

This user license agreement (the AGREEMENT) is an agreement between you (individual or single entity) and MediaMacros, Inc. and Eugene Shoustrov for the included 'ActiveCompanionSet' XTRAS (the SOFTWARE) that are accompanying this AGREEMENT.

The SOFTWARE is the property of Eugene Shoustrov and is protected by copyright laws and international copyright treaties. The SOFTWARE is not sold, it is licensed.

If you accept the terms and conditions of this AGREEMENT, then you are granted the FREE LICENCE.

I. FREE LICENCE

The FREE LICENCE allows using any functionality of the SOFTWARE except for COM Automation objects handled by Automation wrapper of VbScriptXtra and/or OLE objects handled by OLE xtra.

The FREE LICENCE allows using COM Automation handled by Automation wrapper of VbScriptXtra and/or OLE objects handled by OLE xtra with evaluation purposes only.

Any objects or methods that require a license will prompt with an "evaluation" message.

By accepting the FREE LICENCE you have certain rights and obligations as follow:

YOU MAY:

1. Install and use the SOFTWARE (as LICENCE permits) on any computer within your company or home.
2. Make a copy of the SOFTWARE for archival purposes.
3. Distribute an unlimited number of copies of the SOFTWARE with your final runtimes provided that the original package contents stay unchanged including this EULA.

YOU MAY NOT:

1. Sublicense, rent or lease your license
2. Decompile, disassemble, reverse engineer or modify the SOFTWARE or any portion of it, or make any attempt to bypass, unlock, or disable any protective or initialization system on the SOFTWARE.
3. Copy the documentation accompanying the SOFTWARE for use in other software.

II. LIMITED LICENCE

LIMITED LICENSED VERSION

The LIMITED LICENSED VERSION means a Registered Version (using your personal registration number). The LIMITED LICENSE defines a certain set of ProgIds that are allowed to be used with the SOFTWARE.

The LIMITED LICENCE allows using any functionality of the SOFTWARE except for COM Automation objects handled by Automation wrapper of VbScriptXtra and/or OLE objects handled by OLE xtra not covered by the LIMITED LICENCE.

The LIMITED LICENCE allows using COM Automation objects handled by Automation wrapper of VbScriptXtra and OLE objects handled by OLE xtra covered by the LIMITED LICENCE.

The LIMITED LICENCE allows using COM Automation objects handled by Automation wrapper of VbScriptXtra and/or OLE objects handled by OLE xtra not covered by the LIMITED LICENCE with evaluation purposes only.

If you accept the terms and conditions of this AGREEMENT, you have certain rights and obligations as follow:

YOU MAY:

1. Install and use the Registered SOFTWARE on any single computer.

2. Make a copy of the Registered SOFTWARE for archival purposes only.
3. Distribute an unlimited number of copies of the Xtra with your final runtimes provided that the source code is protected and your serial number is not accessible to any 3rd party.

YOU MAY NOT:

1. Copy and distribute the SOFTWARE with an accessible serial number.
2. Sublicense, rent or lease your license
3. Decompile, disassemble, reverse engineer or modify the SOFTWARE or any portion of it, or make any attempt to bypass, unlock, or disable any protective or initialization system on the SOFTWARE.
4. Copy the documentation accompanying the SOFTWARE for use in other software.

III. UNLIMITED LICENCE

UNLIMITED LICENSED VERSION

The UNLIMITED LICENSED VERSION means a Registered Version (using your personal special registration number).

The UNLIMITED LICENCE allows using any functionality of the SOFTWARE.

If you accept the terms and conditions of this AGREEMENT, you have certain rights and obligations as follow:

YOU MAY:

1. Install and use the Registered SOFTWARE on any single computer.
2. Make a copy of the Registered SOFTWARE for archival purposes only.
3. Distribute an unlimited number of copies of the Xtra with your final runtimes provided that the source code is protected and your serial number is not accessible to any 3rd party.

YOU MAY NOT:

1. Copy and distribute the SOFTWARE with an accessible serial number.
2. Sublicense, rent or lease your license
3. Decompile, disassemble, reverse engineer or modify the SOFTWARE or any portion of it, or make any attempt to bypass, unlock, or disable any protective or initialization system on the SOFTWARE.
4. Copy the documentation accompanying the SOFTWARE for use in other software.

WARRANTY DISCLAIMER

The SOFTWARE is supplied "AS IS". MediaMacros, Inc. and Eugene Shoustrov disclaim all warranties, expressed or implied, including, without limitation, the warranties of merchantability and of fitness for any purpose. The user must assume the entire risk of using this SOFTWARE.

DISCLAIMER OF DAMAGES

MediaMacros, Inc. and Eugene Shoustrov assume no liability for damages, direct or consequential, which may result from the use of this SOFTWARE, even if MediaMacros, Inc. and/or Eugene Shoustrov have been advised of the possibility of such damages.

TERM

This license is effective from the date of obtaining or purchasing the SOFTWARE and shall remain in force until terminated. You may terminate the license and this agreement at any time by destroying the SOFTWARE and its documentation, together with all copies in any form that reside on your computer or media.

COPYRIGHT NOTICE:

The Company and/or our Licensors hold valid copyright in the Software. Nothing in this Agreement constitutes a waiver of any rights under U.S. Copyright law or any other federal or state law.

ACKNOWLEDGMENT:

BY USING THIS SOFTWARE YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU ALSO AGREE THAT THIS AGREEMENT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF THE AGREEMENT BETWEEN YOU AND THE COMPANY AND SUPERCEDES ALL PROPOSALS OR PRIOR ENDORSEMENTS, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN YOU AND THE COMPANY OR ANY REPRESENTATIVE OF THE COMPANY RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

All trademarked names mentioned in this document and product are used for editorial purposes only, with no intention of infringing upon the trademarks.

OLE xtra Programmer's Guide

OLE xtra implements custom cast member type; therefore it can be used in the similar way as other visual Director cast members.

Inserting new OLE cast member

Once OLE xtra is placed in Director Xtras folder it adds the menu command for creation new OLE cast members:

```
Insert\XtraMania ::_ActiveCompanionSet ::_OLE object...
```

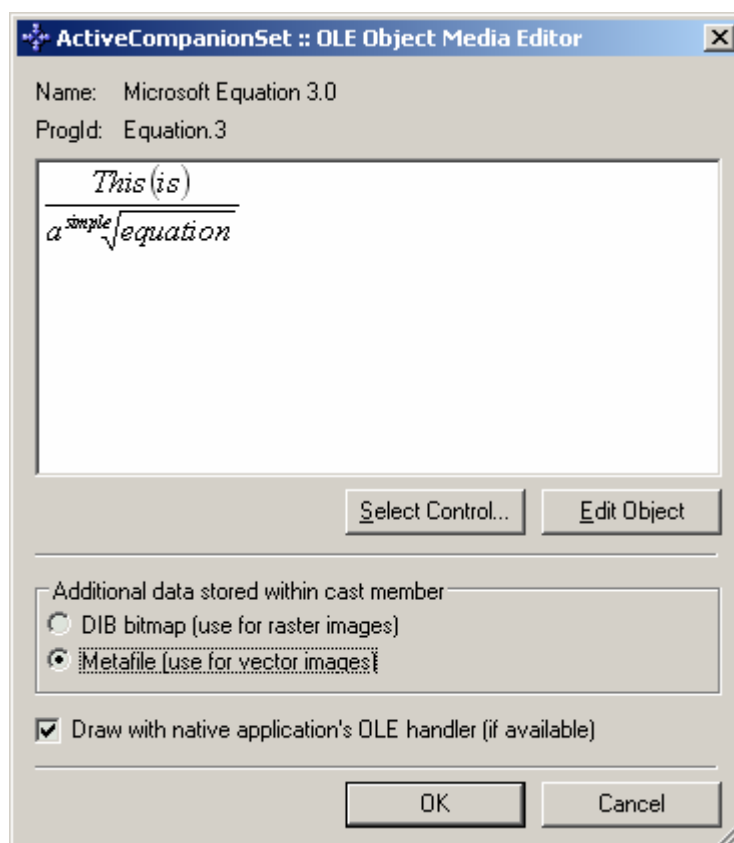
Use this command to insert new empty OLE xtra cast member. The command invokes the OLE xtra media editor described below.

Media editor

Note: OLE xtra's Media Editor is provided as a separate 'OLE xtra UI.dll' file. Make sure you have placed it into the Xtras folder of your Director installation (near with the OLE xtra.x32 file).

Note: You do not need to pack 'OLE xtra UI.dll' file with Projector since it is not used by the xtra while it is running within Projector. It is used only while authoring with Director.

To change existing OLE object cast member double click it or its sprite to invoke media editor dialog.



Additional data stored with cast member

OLE xtra provides three drawing modes for OLE objects. At first, there is a native OLE mode when drawing is performed by native OLE handler provided with native application for the OLE object. Drawing with OLE can only work if native application is installed on the user's system (or at least its OLE handler is registered). This mode is controlled by 'Draw with native application's OLE handler' switch button. If this switch button is checked, then native OLE handler is used if it is available. If this switch button is not checked or OLE handler is not available, then OLE xtra uses static object's image stored together with OLE object data within cast member.

There are two types of this static images stored with asset. It could be either metafile or raster bitmap data (DIB). You can control the type of image stored with asset data by selecting one of 'Additional data stored with cast member' radio buttons.

DIB image is good since you always get the image you saved once. Actually it is the only advantage.

Metafile image is generated by native OLE handler. It is not controlled by OLE xtra. OLE xtra simply saves whatever OLE handler decides to draw, it could be raster data within metafile, it could be some text that relies on fonts installed, it can also be vector curves that do not rely on fonts. That all depend on the particular OLE application.

So it is up to you to choose the type of image stored with the asset.

Scripting operations

OLE xtra fully supports scripting of OLE related operations.

The type of OLE xtra cast member is #OLEobject. You can use it to [create](#) new cast members by Lingo.

OLE xtra allows [inserting OLE object](#) with standard system 'Insert OLE' dialog or programmatically from file or by specifying OLE object ProgId.

OLE xtra provides programming [editing support](#) that allows to open stored OLE object with native application. OLE objects that support COM Automation can be fully controlled in active state by means of VbScriptXtra. OLE xtra can return VbScriptXtra's Automation wrapper for such objects for further scripting. See [the sample](#) with opening and editing Microsoft Word document on the fly.

OLE xtra provides [information](#) about what is happening with the activated object, whether user has changed anything and/or close OLE application. More advanced approach is available via VbScriptXtra's events handling support.

OLE xtra provides scripting control to how OLE object is [appeared](#) on stage, Either with certain type of static image or drawn by the native OLE handler.

Also see [debugging and error handling](#) recommendations for scripting with OLE xtra.

Creating new OLE cast member

The type of the cast member implemented by OLE xtra is #OLEobject. So use the statement below to create a new empty OLE cast member:

```
assetOLE = new( #OLEobject )
```

or

```
assetOLE = new( #OLEobject, member(1) )
```

So, assetOLE is a reference to the newly created OLE xtra's cast member.

Inserting OLE Object

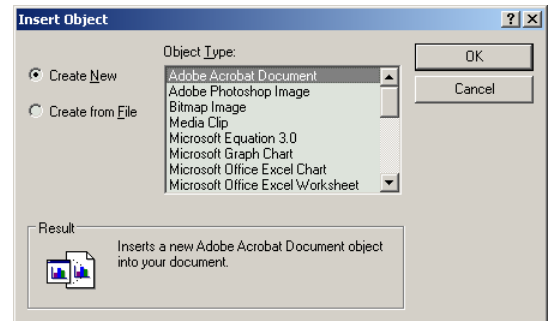
Use `assetOLE.InsertOLE()` method to create an actual OLE object to be stored by the asset. There are several ways to create a new OLE object.

By User Choice

To invoke standard system dialog for inserting OLE objects use the statement below:

```
bSucceeded = assetOLE.InsertOLE()
```

This command will bring the system dialog where user can either choose the type of the new empty OLE object to create or specify a file that should be inserted as OLE object.



If the `bSucceeded` value is true after the calling this method, then OLE object has been created successfully. If it is false, check the [last error information](#) to find out whether it was an OLE specific error or user simply pressed Cancel button in the dialog.

```
bSucceeded = assetOLE.InsertOLE()
if not bSucceeded then
    if assetOLE.Failed then
        -- An error has occurred
        alert assetOLE.LastError
    else
        -- User pressed Cancel
    end if
end if
```

From File

To create a new OLE object from file, use the statement below:

```
bSucceeded = assetOLE.InsertOLE( "d:\Temp\SomeFile.doc" )
```

OLE will automatically find out the native application for the specified file. Check the completion status of the method. For example, it may fail if OLE cannot find the file specified.

Also, it is possible then on different systems OLE will use different native applications for the specified file type. For example if Microsoft Word is not installed, OLE may use packager to simply save the file contents inside the OLE object. You may check which object has been created by this method by checking the [ProgId](#) property of the asset:

```
bSucceeded = assetOLE.InsertOLE( "d:\Temp\SomeFile.doc" )
if bSucceeded then
    put assetOLE.ProgId
end if
```

New Empty Object of Specified Type

To create a new OLE object of the specified type, use the statement below:

```
bSucceeded = assetOLE.InsertOLE( "Word.Document" )
```

The new empty OLE object will be created by the xtra.

Editing OLE object

Once you have an OLE object you can open it with the native application. Use [asset.Open\(\)](#) method to simply open the OLE object with native application.

Note: `asset.Open()` method requires VbScriptXtra to be available, since it uses some of its functionality.

```
-- Create new OLE cast member
assetOLE = new( #OLEobject )

-- Create new OLE object of type Equation
assetOLE.InsertOLE( "Equation" )

-- Open the OLE object with Equation editor
assetOLE.Open()
```

Equation editor will be executed and will show the embedded OLE object. You can check what is happening with the OLE object by checking its [OLEState](#) and [OLEMediaChanged](#) properties.

`OLEState` property allows you to check whether object is still active or user has closed it already.

`OLEMediaChanged` property allows you to check whether user has changed the OLE object data with either Update menu command or by quitting the application (Equation Editor).

Note: OLE xtra keeps the original copy of the OLE object's data with the asset (cast member). When OLE object is activated and updated by the user, asset data stays unchanged unless you explicitly call [asset.Save\(\)](#) method. `OLEMediaChanged` property allows you to know whether OLE object data was changed by a user or not and therefore whether it needs to be saved with cast member or not.

For example see the code below:

```
global assetOLE -- Reference to OLE cast member

on exitFrame me
    if assetOLE.OLEMediaChanged then
        assetOLE.Save()
    end if

    if assetOLE.OLEState <> #Running then
        -- OLE object is closed so we can continue with something else
    end if
end
```

You can close opened object in any time with [asset.Close\(\)](#) method. Closing OLE object means that object passes from #Running state to #Loaded state. Close method has a `bSave` parameter (it is true by default). It allows you to automatically save changes before closing.

Editing OLE object with VbScriptXtra

Some OLE objects may support COM Automation. It allows control them with VbScriptXtra. Once object is opened with native application OLE xtra may get its scripting interface and return it via VbScriptXtra Automation wrapper. Use [asset.GetObject\(\)](#) method to get the Automation wrapper for the activated OLE object.

Note: `asset.GetObject()` method requires VbScriptXtra to be available, since it uses some of its functionality.

```
-- Create new OLE cast member
assetOLE = new( #OLEobject )

-- Create new OLE object of type Word document
assetOLE.InsertOLE( "Word.Document" )

-- Open the OLE object with Microsoft Word
doc = assetOLE.GetObject()

if assetOLE.Failed then
    put assetOLE.LastError
    exit
end if

-- Typing simple message at the beginning of the document
doc.range().InsertAfter( "Editing by Lingo..." )

-- Setting the font style of all text in document
doc.content.bold = #true

-- Now we save changes and close the object
doc = void
assetOLE.Close()
```

In this sample `doc` is usual VbScriptXtra wrapper. Refer to the VbScriptXtra's documentation for more details. More sophisticated applications can use event handling for controlling operations that user performs with the activated OLE document. Also it is possible to control the location of the application window that is used for editing OLE document and so on.

Controlling OLE object appearance

OLE xtra provides three drawing modes for OLE objects. At first, there is a native OLE mode when drawing is performed by native OLE handler provided with native application for the OLE object. Drawing with OLE can only work if native application is installed on the user's system (or at least its OLE handler is registered). This mode is controlled by [`asset.DrawWithOLE`](#) property. If this property is set to `true`, then native OLE handler is used if it is available. If this property is set to `false` or OLE handler is not available, then OLE xtra uses static object's image stored together with OLE object data within cast member.

There are two types of this static images stored with asset. It could be either metafile or raster bitmap data (DIB). You can control the type of image stored with asset data by specifying `symImageType` argument of the [`asset.Save\(\)`](#) method.

DIB image is good since you always get the image you saved once. Actually it is the only advantage.

Metafile image is generated by native OLE handler. It is not controlled by OLE xtra. OLE xtra simply saves whatever OLE handler decides to draw, it could be raster data within metafile, it could be some text that relies on fonts installed, it can also be vector curves that do not rely on fonts. That all depend on the particular OLE application.

So it is up to you to choose the type of image stored with the asset.

Debugging and Errors Handling

There are two main levels of errors related to OLE xtra. They have completely different nature and therefore have to be handled differently.

Lingo Errors

Lingo errors are similar to incorrect Lingo syntax run-time errors. They cause Director to show error alert saying something like "Method or property not found in object" or "One parameter expected". In Projector they might halt script execution etc. These errors usually mean that something is wrong with the programming. Wrong method call syntax is used or something similar to it. OLE xtra might return error codes to Director that make Director to show Lingo error alert box. It happens when OLE xtra discovers the programming error at the Lingo level (wrong syntax, wrong parameters and other compile time evident programming errors).

Programming Errors

This level includes errors that are actually exception conditions. They happen or do not happen depending on particular execution context. They are normal in programming practice and have to be handled programmatically. For example if file operation fails it does not have to worry end-user with Lingo error alert box. Instead developer should check whether operation completed successfully and perform whatever is appropriate.

OLE xtra provides programming errors handling support based on storing status of the last call within every OLE asset object. In other words, every OLE xtra's asset object keeps the error code and description returned by the most recently called method or property. Before returning from the call to any asset object the last error information (if any) is being set by the asset object. Right before calling the next method or property of the asset object the last error information is cleared.

To check the status of the most recent call to the object, use [`asset.Failed`](#) or [`asset.Succeeded`](#) properties. The error message and error code are available via [`asset.LastError`](#) and [`asset.LastErrorCode`](#) properties.

Simple Debugging Mode

Since errors are happening OLE xtra provides debugging modes to simplify debugging process.

In simple debugging mode any asset puts error information into Messages window whenever error occurred. Usually simple debugging mode is useful to detect whether script is executed well or there is a problem somewhere. Error messages usually come from assets but there is no information about the context where error occurred.

To set the simple debugging mode for the particular asset use:

```
member("OLEmember").debugMode = 1
```

Advanced Debugging Mode

Advanced debugging mode allows you to catch error right in Debugger whenever error occurred. In this mode OLE xtra tries to call movie-level handler (it is shared with VbScriptXtra) `VbScriptXtra_DebugEvent(strMes, nCode)`. If there is no such handler, the xtra behaves as in simple debugging mode. This handler may contain any

Lingo statements. Furthermore, you can place a break point inside this handler and use Director's debugging capabilities to view the calling context, variables etc.

Sample movie-level handler for advanced debugging.

```
on VbScriptXtra_DebugEvent strMes, nCode
  put strMes -- Place the break point here
end
```

Debugging mode is kept separately for every OLE xtra asset. Use [DebugMode](#) property to change the debugging mode of the particular asset directly. To set the advanced debugging mode for the particular asset use:

```
member("OLEmember").debugMode = 1
```

OLE xtra Programmer's Reference

OLE xtra implements its own type of Director cast member (asset). OLE xtra asset object (or cast member) can store the OLE object data and its static image in raster or vector form.

OLE xtra implements actor object that can be placed on the stage as Director sprite. OLE xtra actor can show embedded OLE object. It can be drawn either with OLE native handler for the contained object or it can simply display stored static image of the object.

Actor object does not have its own xtra-implemented properties, so all scripting support is implemented on the asset level.

Asset-level

OLE xtra asset provides a scripting control to its contents. It allows [inserting](#) new OLE objects on the fly, [opening](#) them with the native application and then [saving](#) it back to the cast member controlling the [type](#) of the static image saved with the asset.

New OLE objects can be [created](#) either from existing files or empty objects or with system 'Insert OLE object' dialog.

OLE object state can be checked with the [asset.OLEState](#) property.

OLE xtra provides even further scripting control to the contained object by means of VbScriptXtra. If the contained OLE object supports COM Automation it can be activated and controlled with VbScriptXtra. See [asset.GetObject\(\)](#) method.

OLE xtra provides common scripting support similar to VbScriptXtra. It includes [error handling](#) and [debugging](#) support.

Error handling support

Succeeded

Returns true if the most recent call to the asset was successful.

Syntax

```
bResult = asset.Succeeded
```

Return values

True

If the previous call to the asset was successful

False

If the previous call to the asset was not successful. The error code and description are available via [#LastErrorCode](#) and [LastError](#) properties.

Remarks

This property as well as other properties described in this section does not clear the last error flag. It means this property does not affect the last error information for the particular asset object.

Failed

Returns true if the most recent call to the asset has failed.

Syntax

```
bResult = asset.Failed
```

Return values

True

If the previous call to the asset was not successful. The error code and description are available via `LastErrorCode` and `LastError` properties.

False

If the previous call to the wrapper's contents was successful

Remarks

This property as well as other properties described in this section does not clear the last error flag. It means this property does not affect the last error information for the particular asset.

LastErrorCode

Returns the code of the last error (if any) happened while calling the asset.

Syntax

```
nCode = asset.LastErrorCode
```

Return values

Integer

Integer value that indicates the error code of the most recent call to the asset. If the most recent call completed successfully, the error code is 0.

Remarks

This property as well as other properties described in this section does not clear the last error flag. It means this property does not affect the last error information for the particular asset.

Most of error codes are coming from the native application for the contained OLE object.

Other error codes are defined by OLE itself. Here come errors produced by passing incorrect parameters or skipping required parameter etc.

Several error codes are defined by OLE xtra. They could occur if OLE xtra failed to allocate memory for something or similar.

LastError

Returns the description of the last error (if any) happened while calling the asset.

Syntax

```
strErrorMessage = asset.LastError
```

Return values

String

String value that contains the error description of the most recent call to the asset. If the most recent call completed successfully, the error description is empty.

Remarks

This property as well as other properties described in this section does not clear the last error flag. It means this property does not affect the last error information for the particular asset.

Debugging Support

Every OLE xtra asset can detect errors produced while executing OLE operations. Internal OLE xtra errors (memory problems etc) could happen too. Normally these errors could be trapped programmatically by checking asset's last error status after any meaningful call to the asset. See [error handling](#) support properties for more details. To simplify debugging process OLE xtra provides debugging mode.

Simple Debugging Mode

In simple debugging mode any asset object puts error information into Messages window whenever error occurred. Usually simple debugging mode is useful to detect whether script is executed well or there is a problem somewhere. Error messages usually come from wrapped objects but there is no information about the context where error occurred.

Advanced Debugging Mode

Advanced debugging mode allows you to catch error right in Debugger whenever error occurred. In this mode OLE xtra tries to call movie-level handler (it shares the handler with VbScriptXtra) `VbScriptXtra_DebugEvent(strMes, nCode)`. If there is no such handler, the xtra behaves as in simple debugging mode. This handler may contain any Lingo statements. Furthermore, you can place a break point inside this handler and use Director's debugging capabilities to view the calling context, variables etc.

Sample movie-level handler for advanced debugging.

```
on VbScriptXtra_DebugEvent strMes, nCode
    put strMes -- Place the break point here
end
```

Debugging mode is kept separately for every OLE xtra asset object. Debugging mode is not saved with the asset, so use [DebugMode](#) property to change the debugging mode of the particular asset directly.

DebugMode

Sets or gets the debugging mode for the specific asset.

Syntax

```
nDebugMode = asset.DebugMode
asset.DebugMode = nDebugMode
```

Parameters

`nDebugMode` - Integer

Debugging mode for newly created objects. This parameter can be one of the following values.

Value	Meaning
0	No debugging support. Release behavior.
1	Simple debugging. Any error is automatically printed in Messages window.
2	Advanced debugging. When any error is occurred, the xtra calls movie level handler <code>VbScriptXtra_DebugEvent(strMes, nCode)</code> .

Return values

Integer

Integer value that indicates the current debugging mode applied to the wrapper.

Remarks

This property as well as other properties described in this section does not clear the last error flag. It means this property does not affect the last error information for the particular asset.

VbScriptXtra wrapper objects produced by OLE xtra assets get the debugging mode from the asset.

Xtra Specific Properties

Version

This property returns the OLE xtra's version.

Syntax

```
strVersion = asset.Version  
strVersion = asset.Version()
```

Return values

String

Version string in a form of 5 point delimited items: "OLE xtra.1.0.0.4".

The first item is the xtra's name "OLE xtra".

The second item is the major xtra's version.

The third item is the subversion number. It indicates noticeable changes.

The forth item is the minor version number. It indicates minor changes.

The last item is the absolute build number. It is auto incremented with every release build of the xtra.

CLSID

Returns the class Id of the embedded OLE object (if any).

Syntax

```
strClsId = asset.CLSID
```

Return values

String

String value that indicates the CLSID of the embedded object in the registry format.

ProgId

Returns the ProgId of the embedded OLE object (if any).

Syntax

```
strProgId = asset.ProgId
```

Return values

String

String value that indicates the ProgId of the embedded object.

Remarks

This property relies on the registry to determine the ProgId assigned to the OLE object's CLSID. If parent application of OLE object is not installed the property might return empty string even for valid OLE object.

OLEState

Returns the state of the embedded OLE object (if any).

Syntax

```
symObjectState = asset.OLEState
```

Return values

Symbol

Symbol value that indicates the state of the embedded OLE object. It can be one of the following values:

Value	Meaning
#Unloaded	OLE object is not loaded.
#Loaded	OLE object is loaded and controlled by native OLE handler.
#Running	OLE object is opened with the native parent application.

Remarks

If the native OLE object's application is not installed, the object will always stay at #Unloaded state. The xtra will draw such object with stored image of the object.

If object is loaded or running, the xtra may draw object either with native OLE handler or with stored object's image depending on `asset.DrawWithOLE` property.

StoredImageType

Returns the type of object's image stored with asset's media.

Syntax

```
symImageType = asset.StoredImageType
```

Return values

Symbol

Symbol value specifying the type of image that is stored with asset's media. It can be one of the following values.

Value	Meaning
#None	No image is stored with member.

Value	Meaning
#Metafile	OLE object's image is recorded as enhanced metafile.
#DIB	OLE object's image is recorded as raster bitmap image.

Remarks

Metafile image may contain either vector scalable image data or raster bitmap image data. It depends on certain OLE object how it is drawn. OLE xtra allows choosing how to store object's image either as DIB or as Metafile.

DrawWithOLE

Controls the way the asset draws embedded OLE object either with native OLE handler or with object's image stored with asset's media.

Syntax

```
bDrawWithOLE = asset.DrawWithOLE  
asset.DrawWithOLE = bDrawWithOLE
```

Sets or gets

Boolean

Boolean value that indicates whether native OLE handler should be used for drawing object.

Remarks

Note: Metafile view of the OLE object might slightly differ from image drawn by native OLE handler.

Note: Changing the drawing mode may affect how actors are displayed. Especially if stored object is opened with native application and saved within the native application. In this case native OLE handler will display the current object's data, while stored image does not reflect these changes unless [asset.Save\(symImageType\)](#) is called.

OLEMediaChanged

Returns whether the running OLE object was changed and user has accepted changes with either 'Update' menu command or by quitting parent application.

Syntax

```
bMediaChanged = asset.OLEMediaChanged
```

Returns

Boolean

Boolean value that indicates that current embedded OLE object's data differs from the data saved with asset's media.

Remarks

To save changes in the OLE object with asset's media use [asset.Save\(symImageType\)](#) or [asset.Close\(\)](#) methods.

Xtra Specific Methods

Open ()

Method opens the embedded OLE object in its native parent application. Changes of the object made with the parent application could be either ignored or saved back to the cast member with [asset.Save\(symImageType \)](#) method.

Syntax

```
asset.Open( )
```

Return values

VOID

Does not return anything.

Remarks

If [DrawWithOLE](#) property of the asset is set and OLE object is changed by parent application, then asset's sprite might reflect these changes immediately. Data stored with asset can only be changed either with [asset.Save\(symImageType \)](#) method or with Xtra's media editor.

Note: This method relies on some functionality of VbScriptXtra. Therefore it fails if VbScriptXtra is not available.

Method sets the last error information.

GetObject ()

Method opens the embedded OLE object in its native parent application. Then it tries to get IDispatch pointer from the running object to allow controlling it with COM Automation scripting. If successful the instance of Automation wrapper of VbScriptXtra is created to hold the Automation object. This instance is returned by the method.

If OLE object's parent application does not support COM Automation the method returns VOID but behaves as [asset.Open\(\)](#) does.

Syntax

```
objAuto = asset.GetObject( )
```

Return values

Object

If object is created successfully the method returns the new instance of VbScriptXtra wrapper object that holds scripting interface of the running OLE object.

VOID

If the OLE object does not support COM Automation, VOID is returned.

Remarks

Take care with returned object since it keeps the reference to the OLE object keeping it in memory. Make sure to set the variable to VOID to allow OLE object to release its memory.

Note: This method relies on some functionality of VbScriptXtra. Therefore it fails if VbScriptXtra is not available.

Method sets the last error information.

Save(symImageType)

Saves the OLE object data and its image of the requested type into the asset.

Syntax

```
asset.Save()  
asset.Save( Optional Symbol symImageType )
```

Parameters

symImageType

Symbol value specifying the type of image to be stored with asset.

If image type is not specified then the current asset's image type is used. Otherwise it can be one of the following values.

Value	Meaning
#None	No image is stored with member.
#Metafile	OLE object's image is recorded as enhanced metafile.
#DIB	OLE object's image is recorded as raster bitmap image.

Return values

VOID

Does not return anything.

Remarks

This method is the only Lingo way to update OLE xtra asset's media data.

Note: If [DrawWithOLE](#) property of the asset is set and OLE object is changed by parent application, then asset's sprite might reflect these changes immediately. Data stored with asset can only be changed either with this method or with Xtra's media editor.

Note: Metafile image may contain either vector scalable image data or raster bitmap image data. It depends on certain OLE object how it is drawn. OLE xtra allows choosing how to store object's image either as DIB or as Metafile.

Method sets the last error information.

Close(bSave)

Saves and closes running OLE object. Closing means that object goes from Running state into Loaded state.

Syntax

```
asset.Close()  
asset.Close( Optional Boolean bSave )
```

Parameters

bSave

Boolean value that indicates whether object has to be saved. By default it is set to true. If bSave is set to false object is closed and changes made to the object are discarded.

Return values

VOID

Does not return anything.

Remarks

This method closes the running OLE object (if it is running). Then internally calls Save method using the current asset's type of object's image.

Method sets the last error information.

LoadOLEObject ()

Method tries to load an instance of embedded OLE object. This operation succeeds only if OLE handler is available for this particular kind of objects. Normally OLE handler is installed with the parent object's application.

Syntax

```
bSucceeded = asset.LoadOLEObject ( )
```

Return values

Integer

Integer value that indicates whether operation has succeeded.

Remarks

If [DrawWithOLE](#) property of the asset is set, it will automatically try to load OLE object when any of its sprites are needed to draw itself. Most of scripting methods try to load OLE object too.

There are rare cases when this method might be used.

Method sets the last error information.

UnloadOLEObject ()

Method unloads the instance of embedded OLE object.

Syntax

```
bSucceeded = asset.UnloadOLEObject ( )
```

Return values

VOID

Does not return anything.

Remarks

This method allows simulating xtra's behavior in case when OLE handler is not available for the embedded object. In this case asset will use stored image data to draw object.

Method sets the last error information.

InsertOLE()

Initializes the asset with new OLE object. New OLE object could be either specified by its ProgId or source file, or it could be selected by user with system 'Insert OLE object' dialog box.

Syntax

```
bSucceeded = asset.InsertOLE( Optional String strSource )
```

Parameters

`strSource`

String value that indicates the object to be inserted. If `strSource` is not specified, the system 'Insert OLE object' window will be invoked. It allows user to choose either which OLE object to insert or from which file to create OLE object.

If `strSource` is specified it could be one of the following values:

Value	Meaning
" {CLSID} "	New OLE object by the specified CLSID in registry format.
"ProgId"	New OLE object by its ProgId (i.e. "Word.Document")
"path-to-a-file"	New OLE object from the specified file.

Return values

Integer

Integer value that indicates whether operation has succeeded. If user has cancelled the Insert OLE dialog, then the method returns 0 and sets the [Succeeded](#) status to true.

Remarks

This method discards the current media of the asset only if the new OLE object is successfully created.