ActiveX xtra
Version 1.0

www.xtramania.com

ActiveX xtra

About ActiveX xtra

ActiveX xtra extends the Macromedia Director's Lingo functionality with capability to handle ActiveX visual controls.

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

ActiveX 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

ActiveX 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, ActiveX xtra and ObjectBrowserXtra.

What is the Difference

Macromedia Director ships with its own ActiveX xtra. ActiveCompanionSet provides even better support for ActiveX visual controls due to advanced scripting support by means of VbScriptXtra. Below is the list of key features provided by ActiveCompanionSet for visual ActiveX controls.

- Advanced scripting support. Supports almost all methods and properties including those that uses data types other then simply strings and numbers.
- Events handling allows using either behavior or parent script instance for handling events fired by visual ActiveX control.
- Events, methods and properties of ActiveX control can be viewed with ObjectBrowser xtra.

'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 LICENSE.

I. FREE LICENSE

The FREE LICENSE allows using any functionality of the SOFTWARE except Automation and Typed Instance wrappers of VbScriptXtra.

The FREE LICENSE allows using Automation and Typed Instance wrappers of VbScriptXtra with evaluation purposes only.

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

YOU MAY:

Install and use the SOFTWARE (as LICENSE permits) on any computer within your company or home.

Make a copy of the SOFTWARE for archival purposes.

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:

Sublicense, rent or lease your license

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.

Copy the documentation accompanying the SOFTWARE for use in other software.

II. LIMITED LICENSE

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 LICENSE allows using any functionality of the SOFTWARE except for Automation and Typed Instance wrappers of VbScriptXtra for objects with ProgIds not covered by the LIMITED LICENSE.

The LIMITED LICENSE allows using Automation and Typed Instance wrappers of VbScriptXtra for objects with ProgIds covered by the LIMITED LICENSE.

The LIMITED LICENSE allows using Automation and Typed Instance wrappers of VbScriptXtra for objects with ProgIds not covered by the LIMITED LICENSE with evaluation purposes only.

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

YOU MAY:

Install and use the Registered SOFTWARE on any single computer.

Make a copy of the Registered SOFTWARE for archival purposes only.

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:

Copy and distribute the SOFTWARE with an accessible serial number.

Sublicense, rent or lease your license

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.

Copy the documentation accompanying the SOFTWARE for use in other software.

III. UNLIMITED/UNLIMITED PLUS LICENSE

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

The UNLIMITED LICENSE does not imply any restrictions on ProgIds being used with the SOFTWARE. This license allows using any functionality of the SOFTWARE except for Typed Instance wrappers of VbScriptXtra.

The UNLIMITED PLUS LICENSE does not imply any restrictions on ProgIds being used with the SOFTWARE. This license allows using any functionality of the SOFTWARE including Typed Instance wrappers of VbScriptXtra.

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

YOU MAY:

Install and use the Registered SOFTWARE on any single computer.

Make a copy of the Registered SOFTWARE for archival purposes only.

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:

Copy and distribute the SOFTWARE with an accessible serial number.

Sublicense, rent or lease your license

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.

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.

ActiveX xtra Programmer's Guide

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

Inserting new ActiveX cast member

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

`Insert\XtraMania :: ActiveCompanionSet :: ActiveX control...`

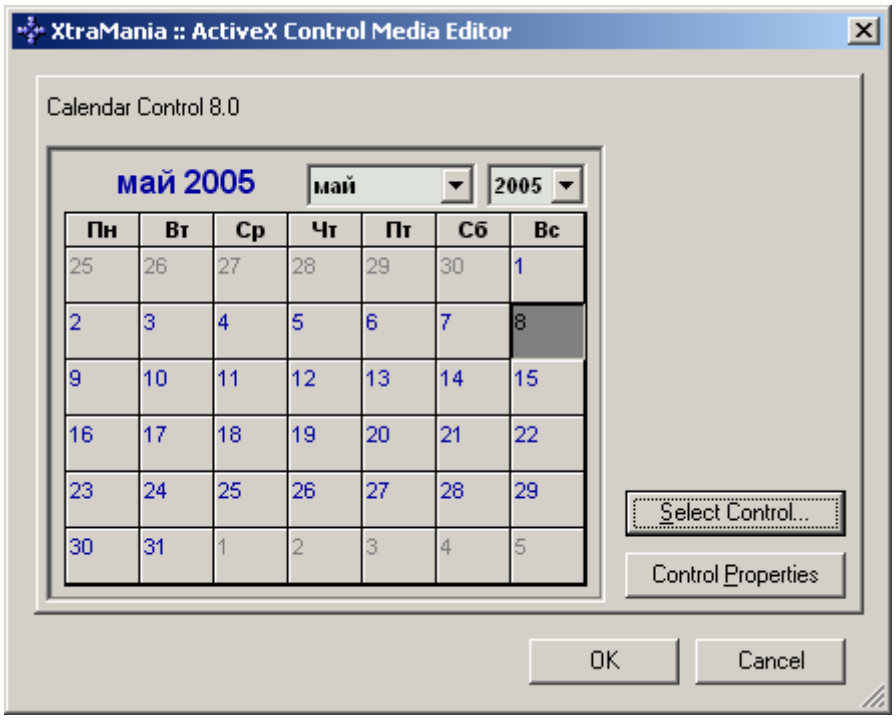
Use this command to insert new empty ActiveX xtra cast member. The command invokes the ActiveX xtra media editor described below.

Media editor

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

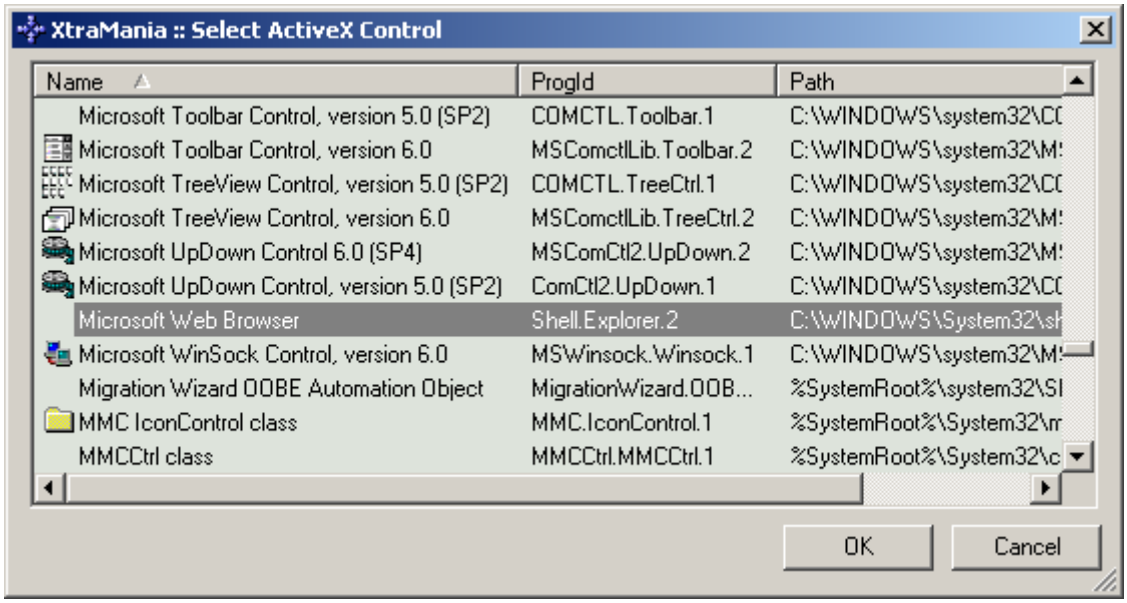
Note: You do not need to pack '`ActiveX xtra UI.dll`' file with Projector since it is not used by the xtra while it is running within Projector. It is used only while authoring with Director.

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



`Control properties` button invokes the control's properties editor dialog box.

Press `Select control` button to choose ActiveX control. The dialog window with available ActiveX controls will appear.



Scripting operations

The type of ActiveX xtra cast member is `#ActiveXControl`. You can use it to create new cast members by Lingo and then specify the ActiveX control for the member.

Scripting ActiveX control and processing events are handled by VbScriptXtra Automation wrapper.

Also see debugging and error handling recommendations for scripting with ActiveX xtra.

Creating new ActiveX cast member

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

```
assetActiveX = new( #ActiveXControl )
```

or

```
assetActiveX = new( #ActiveXControl, member(1) )
```

So, `assetActiveX` is a reference to the newly created ActiveX xtra's cast member. Use `asset.InsertActiveX()` to programmatically create specific ActiveX control by either ProgId or CLSID.

```
assetActiveX.InsertActiveX( "Shell.Explorer" )
```

Scripting ActiveX control with VbScriptXtra

ActiveX controls usually support COM Automation. It allows scripting them with VbScriptXtra. Once control is running on stage as a sprite ActiveX xtra may get its scripting interface and return it via VbScriptXtra Automation wrapper. Use `sprite.GetObject()` method to get the Automation wrapper for the running ActiveX control.

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

Some ActiveX controls may fire events for handling user actions or other things. Use `EventsHandler` property of the VbScriptXtra wrapper returned by `sprite.GetObject()` method to set the handler for these events. Events could be handled by either parent script instance or sprite's behavior.

Below is the code of behavior that may be placed on Internet Explorer ActiveX sprite. It prevents user from opening new IE windows on Shift + Click on the URL. Instead it makes the control itself to browse to the requested page.

```
property spriteNum
property mControl

on beginSprite me
  sprite(spriteNum).debugMode = true

  -- Trying to get Automation object for ActiveX control
  mControl = sprite(spriteNum).GetObject()

  -- Set me to be the event handler for ActiveX control
  mControl.EventsHandler = me

  mControl.Navigate("www.xtramania.com")
end

on endSprite me
  -- Make sure to clear event handler
  mControl.EventsHandler = VOID
end

-- Generic events handler for ActiveX control
on IncomingEvent me, event, args
  case event of
    #NewWindow3:
      -- We do not wnat user to be able opening new IE window,
      -- so we cancel operation and simply make the
      -- current IE to navigate to the requested page
      args[#Cancel] = #true

      put "Navigate to: " & args[#bstrUrl]
      mControl .Navigate( args[#bstrUrl] )

    otherwise
      -- Put other events to Messages window just to look at them
      put event,args
   end case
end
```

In this sample `mControl` is usual `VBScriptXtra` wrapper. Refer to the VbScriptXtra's documentation for more details.

Use `mControl.Interface()` method to invoke ObjectBrowser xtra with detailed description of what you can do with it.

Debugging and Errors Handling

There are two main levels of errors related to ActiveX 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. ActiveX xtra might return error codes to Director that make Director to show Lingo error alert box. It happens when ActiveX 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.

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

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

Simple Debugging Mode

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

In simple debugging mode any asset or actor 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. These error messages contain no information about the context where error occurred.

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

```
member("ActiveXmember").debugMode = 1
sprite(spriteNum).debugMode = 1
```

Advanced Debugging Mode

Advanced debugging mode allows you to catch error right in Debugger whenever error occurred. In this mode ActiveX 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 ActiveX xtra asset and actor. Use DebugMode property to change the debugging mode of the particular object directly. To set the advanced debugging mode for the particular object use:

```
member("ActiveXmember").debugMode = 2
sprite(spriteNum).debugMode = 2
```

ActiveX xtra Programmer's Reference

ActiveX xtra implements its own type of Director cast member (asset). ActiveX xtra asset object (or cast member) keeps the reference to the particular type of ActiveX control as well as its initialization properties.

ActiveX xtra implements actor object that can be placed on the stage as Director sprite. ActiveX xtra actor can show visual ActiveX control as a Director always on top sprite.

Most of the scripting support is implemented by means of VbScriptXtra. ActiveX xtra actor provides only a few identification properties and method providing access to the COM Automation interface of the ActiveX control.

Common properties for assets and actors

ActiveX 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.

`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 ActiveX xtra asset can detect errors produced while executing ActiveX operations. Internal ActiveX 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 ActiveX 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 ActiveX 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 ActiveX xtra asset or actor 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 `VbScriptXtra_DebugEvent( strMes, nCode )`.

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.

ActiveX xtra actors produced by the ActiveX xtra assets inherit asset's debugging mode.

VbScriptXtra wrapper objects produced by ActiveX xtra actors get the debugging mode from the asset.

Asset-level

ActiveX xtra asset provides a scripting identification control to its contents.

ActiveX xtra provides common scripting support similar to VbScriptXtra. It includes error handling and debugging support.

Xtra Specific Properties

Version

This property returns the ActiveX xtra's version.

Syntax

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

Return values

`String`

Version string in a form of 5 point delimited items: "ActiveX xtra.1.0.0.3".

The first item is the xtra's name "ActiveX 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 wrapped ActiveX control (if any).

Syntax

```
strClsId = asset.CLSID
```

Return values

`String`

String value that indicates the CLSID of the control type in the registry format.

Remarks

The same property exists at the actor level.

ProgId

Returns the ProgId of the wrapped ActiveX control (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 ActiveX control's CLSID. If ActiveX control is not installed the property might return empty string even for valid ActiveX object.

The same property exists at the actor level.

Xtra Specific Methods

`InsertActiveX()`

Initializes the asset with new ActiveX control object. New ActiveX object could be either specified by its ProgId or CLSID.

Syntax

```
bSucceeded = asset.InsertActiveX( String strSource )
```

Parameters

`strSource`

String value that indicates the object to be inserted. 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. `"Shell.Explorer"`)

Return values

`Integer`

Integer value that indicates whether operation has succeeded.

Remarks

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

Actor-level

ActiveX xtra actors represent Director sprites. ActiveX xtra actor provides a scripting control to its contents. Once control is running on stage as a sprite ActiveX xtra may get its scripting interface and return it via VbScriptXtra Automation wrapper. Use `sprite.GetObject()` method to get the Automation wrapper for the running ActiveX control.

Use `sprite.Focus` property to handle whether and when ActiveX control gets the keyboard input.

ActiveX xtra provides common scripting support similar to VbScriptXtra. It includes error handling and debugging support.

Xtra Specific Methods

`GetObject()`

Method tries to get IDispatch pointer (scripting interface) from the running ActiveX control 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 ActiveX control does not support COM Automation the method returns `VOID`.

Syntax

```
objAuto = sprite(spriteNum).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 ActiveX object.

`VOID`

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

Remarks

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

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

Xtra Specific Properties

`CLSID`

Returns the class Id of the wrapped ActiveX control (if any).

Syntax

```
strClsId = actor.CLSID
```

Return values

`String`

String value that indicates the CLSID of the control type in the registry format.

Remarks

The same property exists at the asset level.

ProgId

Returns the ProgId of the wrapped ActiveX control (if any).

Syntax

```
strProgId = actor.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 ActiveX control's CLSID. If ActiveX control is not installed the property might return empty string even for valid ActiveX object.

The same property exists at the asset level.

Focus

Sets or gets whether the wrapped ActiveX control's window has a keyboard focus.

Syntax

```
bFocused = actor.Focus
actor.Focus = bFocused
```

Parameters

`bFocused - Boolean`

Setting the property allows controlling whether the Stage or the ActiveX control's window gets the keyboard input. This parameter can be one of the following values.

Value	Meaning
`False`	The stage gets the keyboard input.
`True`	The ActiveX control gets the keyboard input.

Return values

`Boolean`

Boolean value that indicates whether the ActiveX control's window has a focus.

Remarks

This property allows handling keyboard navigation between Director sprites and another ActiveX controls.

ActiveX xtra automatically sets the keyboard focus on control's window when its sprite is set as an input target by Director. It can happen either with Director auto tab handling or with Lingo by setting `the keyboardFocusSprite` property.

Note: some controls may not provide interface for accessing its window. In this case this functionality may not work as expected.

Sample

The behavior below demonstrates how to handle keyboard focus with ActiveX sprite. It is supposed to be placed on ActiveX xtra's sprite with a control that provides `KeyDown` event.

```
property spriteNum
property mControl

on beginSprite me
  sprite(spriteNum).debugMode = true

  mControl = sprite(spriteNum).GetObject()
  mControl.EventsHandler = me
end

on endSprite me
  mControl.EventsHandler = VOID
end

on IncomingEvent me, event, args
  put event,args
  if event = #KeyDown then
    me.ActiveXKeyDown( args )
  end if
end

on ActiveXKeyDown me, args
  if args[#KeyCode] = 9 and args[#Shift] = 0 then
    sprite(spriteNum).Focus = false
    the keyboardFocusSprite = spriteNum + 1 -- '+1' is for simplicity
  else if args[#KeyCode] = 9 and args[#Shift] = 1 then
    sprite(spriteNum).Focus = false
    the keyboardFocusSprite = spriteNum - 1 -- '-1' is for simplicity
  end if
end
```

Note: it is up to control whether or not to provide `keyDown` event. This event is sent by VbScriptXtra's event handling (the original event is fired by ActiveX control).

Take with direct handling of `keyDown` event since if the focus is not set to the control's window and `the keyboardFocusSprite` is set to the ActiveX xtra's sprite, then `keyDown` may work as normal Lingo `on keyDown` handler. `args` parameter will be set to `VOID` in this case.

HWND

Returns the window handle (HWND) of the wrapped ActiveX control (if any).

Syntax

```
hWnd = actor.HWND
```

Return values

`Integer`

value that indicates the window handle of the ActiveX control.

Remarks

Some controls may not provide interface for accessing its window. In this case this property will return VOID. Even so, ActiveX control may provide its own HWND property to allow access to its window. Check its interface with ObjectBrowser.