VbScriptXtra Documentation

www.xtramania.com

VbScriptXtra documentation 4

VbScriptXtra usage 4

About VbScriptXtra 4

What is VbScriptXtra? 4
What is COM Automation? 4
What can I do with VbScriptXtra? 4

Common guidelines 6

VbScriptXtra emulates VB syntax as close as possible 6
VbScriptXtra provides wrapper instance for every automation object reference 6
Use `CreateObject` to 'automate' something 6
Using VbScriptXtra wrapper instance 7
What can I do with Word application? 7

Mapping between Lingo data types and Automation data types 8

Debugging your scripts using VbScriptXtra 8

Autodocumentation feature 9

Getting interface description of Automation objects 9
Using ObjectBrowser to view available methods and properties 9

Frequently Asked Questions about VbScriptXtra 10

Can I use DAO or ADO with VbScriptXtra? 10

Which ProgIds can I use with VbScriptXtra? 11

VbScriptXtra Reference 13

Version history 13

Technical details 17

Default object's property 17
Default object's method (or indexed property) 17
Events 17
Named method's arguments 18
Optional and missing method's arguments 18
Passing parameters by reference 18
Arrays as method's arguments 18
Arrays as a return value 18
Lingo syntax issues 19
'Underscore handling' 20

Xtra-level functions, provided by VbScriptXtra: 21

`xtra "VbScriptXtra".CreateObject(strProgId)` 21
`xtra "VbScriptXtra".GetObject(strProgId)` 22
`xtra "VbScriptXtra".Init(bDebugMode)` 22
`xtra "VbScriptXtra".Version()` 23

Properties and methods provided by VbScriptXtra automation object wrapper 24

`objAuto.CodePage` 25
`objAuto.DebugMode` 28
`objAuto.EventsHandler` 28
`objAuto.Failed` 32
`objAuto.LastError` 32
`objAuto.Succeeded` 33
`enums(objAuto)` 33
`interface(objAuto)` 34

VbScriptXtra automation object wrapper - automatic type casting 35

Mapping Lingo types to COM Automation types 35
Mapping COM Automation types to Lingo types 37

VbScriptXtra Samples 39

Open ADO Recordset sample script 40

Where to find more info about ADO? 41

"Hello World!" for Excel = "Hello Excel!" 42

Where to find more info about Excel? 42

Silent navigation to url 44

Where to find more info about Internet Explorer? 45

Retrieving contacts from MSN Messenger 46

Where to find more info about Messenger? 46

"Hello World!" for Word = "Hello Word!" 47

Where to find more info about Word? 47

Contacts 49

VbScriptXtra documentation

VbScriptXtra usage

About VbScriptXtra

Several words about what VbScriptXtra is, what COM Automation technology is and several thoughts about where to use VbScriptXtra.

What is VbScriptXtra?

VbScriptXtra is an extension of Macromedia Director of type 'scripting xtra'. It extends the capabilities of Lingo (Director's scripting language) with ability to access external applications or system components using COM Automation. VbScriptXtra allows Director developers to fully automate such applications like: Microsoft Word, Excel, Power Point, Access and other Office components, Microsoft Visual Source Safe, Microsoft Internet Explorer, ADO, DAO and other system components, custom components created with Visual Basic and virtually any application which supports COM Automation technology.

What is COM Automation?

Automation is a COM-based technology that allows developers to automate frequent or complicated tasks using one or several applications. Automation allows applications to expose their functionality to scripting languages and other interpreted languages. Automation allows scripting languages to control applications' functionality. Automation was designed primarily for Visual Basic developers, but with VbScriptXtra it is available to Director developers. Automation is available on Windows 95/98/ME and Windows NT/2000/XP operating systems.

What can I do with VbScriptXtra?

Have you ever write macros in Word to automate some routine operation? Not yet? With VbScriptXtra you may write Word macros using Lingo and execute them from projector. For example, your projector may silently run Word application and while it is invisible to the user, open a document, fill it with appropriate info, make corresponding character formatting and then show Word document to the user or print it.

Microsoft products usually have good spell checking capabilities. Why not use Word to spell check your text cast members? Simple script scenario: iterate through text cast members, put the text to the Word, invoke spell checking, save results back into cast member.

You may want to open web site from projector. For sure you may use `GoToNetPage` Lingo command, but you do not know anything about what happens next. With VbScriptXtra you may fully control Internet Explorer: its visibility, window location, command bars, full screen state, readiness state and much more.

If you want projector to show some table like information, you may run Excel and fill it with appropriate information. You may fully control everything in Excel and close it when you do not need it anymore.

All this features are available to Visual Basic users including any VBA, VbScript, or VB. With VbScriptXtra it is available to Director.

Common guidelines

Basic things you have to know to start using VbScriptXtra.

VbScriptXtra emulates VB syntax as close as possible

VbScriptXtra extends the functionality of Lingo to allow you to use Lingo to automate applications or components. Automation was designed mostly for Visual Basic. VB syntax is native for Automation. VbScriptXtra tries to emulate Vb syntax as close as possible with Lingo. See more details about differences between real VB syntax and Lingo syntax using VbScriptXtra in reference section.

VbScriptXtra provides wrapper instance for every automation object reference

VbScriptXtra very extensively uses Lingo's dot syntax to allow cascading access to automation object's methods and properties, which may also return other automation objects. VbScriptXtra provides its own wrapper instance for every automation object reference. You may think about such wrappers as just Lingo value of some special type. Any method or property access of the automation object is implemented through this wrapper.

`CreateObject` method is the main starting point while using VbScriptXtra. Like its Visual Basic analogue, this method creates a new instance of the automation object and returns it as a Lingo value. This Lingo value is provided by VbScriptXtra to allow direct access to the object's methods and properties.

Use `CreateObject` to 'automate' something

`CreateObject` is implemented by VbScriptXtra as xtra-level method. So it can be called only in xtra context. This is done intentionally to avoid possible conflicts with other xtras that may use the same method's name. So Lingo allows you to either place xtra reference as the first argument of the method or use dot syntax. Following lines does the same:
`objWord=CreateObject(xtra "VbScriptXtra","Word.Application")` or
`objWord=xtra("VbScriptXtra").CreateObject("Word.Application")`

The most important argument of the `CreateObject` method (the last one) is the `ProgId` of the automation object. For example, Microsoft Word provides two possible `ProgIds` allowed to be used from scripting `'Word.Application'` and `'Word.Document'`. Other applications may define their own `ProgIds`.

The returned Lingo value is actually a VbScriptXtra instance, which wraps an automation object. This VbScriptXtra wrapper passes onto wrapped automation object the corresponding method and property calls and makes all necessary type casting. For example, after initial creation of Word automation object you will not see the Word window, since it is run as hidden application. Using `objWord` Lingo value (wrapper

instance) you may control its visibility. Use `objWord.visible=true` and Word window becomes visible.

Using VbScriptXtra wrapper instance

VbScriptXtra wrapper provides advanced cascading properties and methods access. Microsoft Word application implements rather large object model with `application` object on the top. Word application implements such objects like Documents collection, Document object, Text range, Style object, Search object etc. Using `objWord` as an entry point you may open or create a new document: `objWord.Documents.Add()` Then you may get the name of the newly created document: `put objWord.Documents(1).Name put objWord.ActiveDocument.Name`

To simplify your further code you may assign a new variable with a reference to the active Word document: `objDoc=objWord.ActiveDocument put objDoc.Name`

What can I do with Word application?

Well, Microsoft Word implements very large object model since everything you can do by hands you can do by script. You may want to know where to find information about how to automate Word or Excel or PowerPoint or Access. Refer to the help system of the corresponding application. See 'Visual Basic for Application' topic to find all possible information about automation.

Sometimes, it is difficult to find documentation, sometimes it is easier to use autodocumentation feature of VbScriptXtra. Just use: `put interface(objWord)` to see what you can do with objWord instance. The same way you may go further and see what you can do with `objWord.Documents` collection: `put interface(objWord.Documents)`

Many properties and methods of different automation objects often use named constant values (so called enumerations). To see the list of all known enumerations use: `put enums(objWord)` VbScriptXtra provides you a way to use such constant values using their names. Just use corresponding VbScriptXtra wrapper. For example to get the value of any enumeration constant known to Word use:

```
put objWord.SomeEnumerationHere
```

VbScriptXtra provides an additional authoring component: Object browser. It is a special component, which allows you to browse through different type descriptions of automation objects installed on your system. It is available for free at www.xtramania.com. See Autodocumentation feature below for more information.

Mapping between Lingo data types and Automation data types

VbScriptXtra performs automatic type casting to correctly transfer data between Lingo and Automation object and vice versa. VbScriptXtra implicitly performs typecasting operations during processing Lingo method arguments, returning values, and property values. Automation and Visual Basic supports rather large amount of data types. Lingo has its own Director specific data types. So, VbScriptXtra may not find suitable conversion in all cases, although it provides conversion in the most cases. If VbScriptXtra does not know how to convert the value it will report an error. See detailed description which types are mapped to which types in Reference section.

Download free BinaryXtra, which extends the type casting capabilities of VbScriptXtra allowing it to handle BLOB data.

Debugging your scripts using VbScriptXtra

Every VbScriptXtra wrapper instance has internal last error flag and error description. The last error flag is cleared before any access to the Automation object. If property access or method call failed or another error happened, this flag is raised. So you may detect whether last call completed successfully. Use `object.Failed` or `object.Succeeded` properties to check whether the last call was successful. If an error happened you may see its description using `object.lastError` property.

Also you may adjust VbScriptXtra wrapper objects to output its `lastError` directly to the Message window every time error happens. Set `object.DebugMode` to true (1) to do this. By default, VbScriptXtra wrappers created by `CreateObject` call inherit the xtra's default value for debugMode. You may change this default with `Init` xtra-level method.

Note: While you are just investigating automation capabilities it is better to set Debug Mode by default, to ensure you always know if something goes wrong. It might be important while using cascading properties, since every property returning Automation Object will be wrapped by a new instance of VbScriptXtra wrapper, which has its own error flag and error description. So, you may skip useful error description from wrappers created on the fly, if you do not set Debug Mode.

Autodocumentation feature

Getting interface description of Automation objects

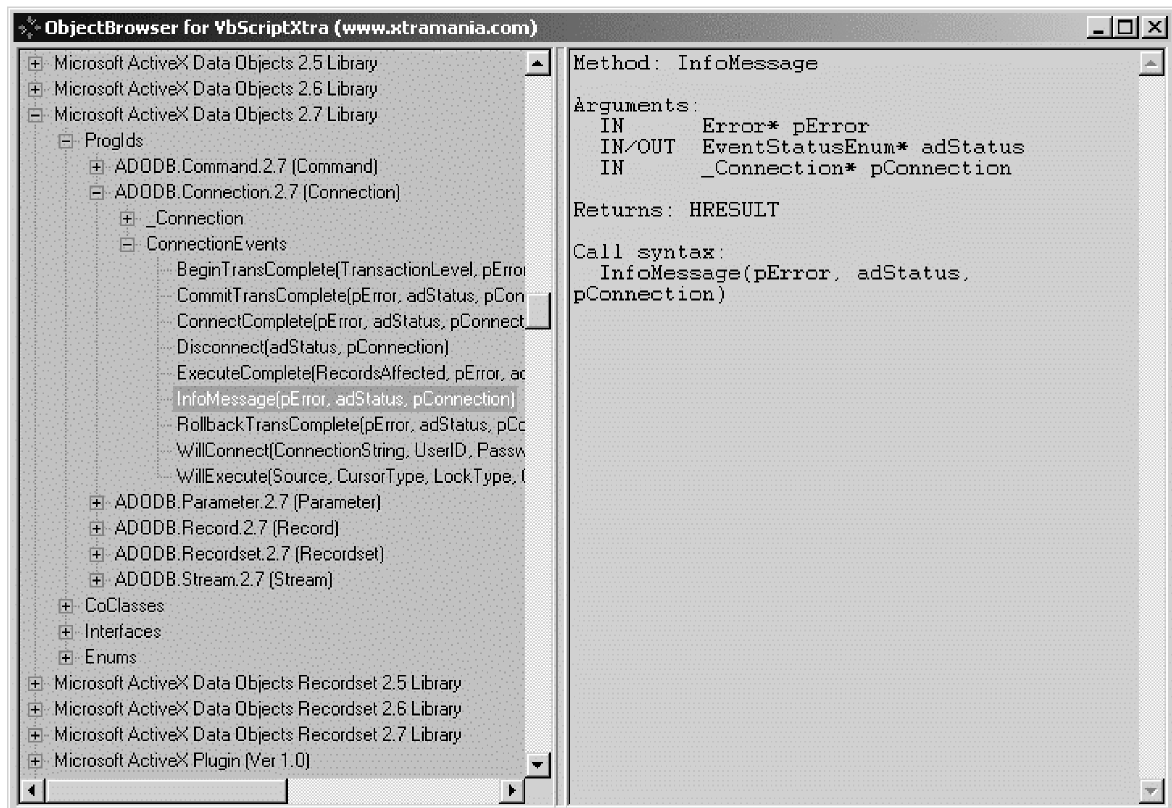
VbScriptXtra provides a built in autodocumentation feature. Once you have valid Automation Object wrapper, you may use usual Lingo `put objAuto` command to see the name of the wrapped Automation interface. Also you may get the detailed description of this interface including detailed information about properties and methods it provides. Use: `put interface(objAuto)` to see what you can do with objAuto instance.

Many properties and methods of different automation objects often use named constant values (so called enumerations). To see the list of all known enumerations use: `put enums(objAuto)` VbScriptXtra provides you a way to use such constant values using their names. Just use corresponding VbScriptXtra wrapper. For example to get the value of any enumeration constant known to objAuto use:

```
put objAuto.SomeEnumerationHere
```

Using ObjectBrowser to view available methods and properties

Autodocumentation companion is available for VbScriptXtra at Downloads section. It provides an information about available scriptable components right from Macromedia Director. See more info about ObjectBrowser in Documentation section.



Frequently Asked Questions about VbScriptXtra

Can I use DAO or ADO with VbScriptXtra?

Yes, it is possible. VbScriptXtra works well with Microsoft Data Access Objects (DAO) or Microsoft ActiveX Data Objects (ADO).

Unlike ADOxtra, which is directly targeted to providing ADO in Director, VbScriptXtra provides a way to use all scriptable objects (COM Automation). So, VbScriptXtra documentation does not provide a specific description of ADO itself, although it provides several samples. Refer to ADO documentation files shipped with MDAC (Microsoft Data Access Components).

You may use ObjectBrowser autodocumentation companion to invoke a help file for ADO or DAO. Make sure you have put ObjectBrowser xtra into Director's xtras folder. Select Xtras\VbScriptXtra\Object Browser... to invoke ObjectBrowser window. To invoke a help file for ADO find 'Microsoft ActiveX Data Objects 2.5 Library' item, right click it and choose 'Invoke type library help'.

To invoke a help file for DAO find 'Microsoft DAO 2.5/3.5 Compatibility Library' item, right click it and choose 'Invoke type library help'.

Which ProgIds can I use with VbScriptXtra?

COM Automation is open well-documented technology. Potentially any application may support COM Automation technology.

Below is a list of ProgIds to automate really useful software, known to the author:

`"Word.Application"` - Microsoft Word
`"Word.Document"` - Microsoft Word document
`"Word.Picture"` - Microsoft Word picture document

`"Excel.Application"` - Microsoft Excel
`"Excel.Sheet"` - Microsoft Excel sheet
`"Excel.Chart"` - Microsoft Excel chart

`"MSGraph.Application"` - Microsoft Graph application
`"MSGraph.Chart"` - Microsoft Graph chart

`"PowerPoint.Application"` - Microsoft PowerPoint
`"PowerPoint.Slide"` - Microsoft PowerPoint slide

`"Access.Application"` - Microsoft Access

`"InternetExplorer.Application"` - Microsoft Internet Explorer

`"NetMeeting.Application"` - Microsoft NetMeeting

`"Agent.Control"` - Microsoft Agent

`"ADODB.Connection"` - ADO Connection object
`"ADODB.Recordset"` - ADO Recordset object
`"ADODB.Record"` - ADO Record object
`"ADODB.Command"` - ADO Command object
`"ADODB.Parameter"` - ADO Parameter object
`"ADODB.Stream"` - ADO Stream object

`"ADOX.Catalog"` - ADO Catalog object
`"ADOX.Group"` - ADO Group object
`"ADOX.User"` - ADO User object
`"ADOX.Table"` - ADO Table object
`"ADOX.Column"` - ADO Column object
`"ADOX.Key"` - ADO Key object
`"ADOX.Index"` - ADO Index object

`"DAO.DBEngine.35"` - DAO DbEngine object
`"DAO.Field.35"` - DAO Field object
`"DAO.Group.35"` - DAO Group object

`"DAO.Index.35"` - DAO Index object
`"DAO.QueryDef.35"` - DAO QueryDef object
`"DAO.Relation.35"` - DAO Relation object
`"DAO.TableDef.35"` - DAO TableDef object
`"DAO.User.35"` - DAO User object

`"Photoshop.Application"` - Adobe Photoshop application
`"ImageReady.Application"` - Adobe ImageReady application

`"MSComDlg.CommonDialog"` - Common dialog control

`"Shell.Application"` - System Shell

`"Scripting.FileSystemObject"` - Basic file system object

`"WScript.Shell"` - Windows Scripting Host Shell object
`"WScript.Network"` - Windows Scripting Host Network object

Object browser extension is available at Downloads. It is free. Download it to visually investigate what you can do with VbScriptXtra. More information about ObjectBrowserXtra is available in Documentation.

VbScriptXtra Reference

Version history

information about how VbScriptXtra is growing.

May 5th, 2002

Version 1.02.004 Build 21
Added support for custom Unicode text conversion. Director uses MBCS (Multibyte Character Set). Every character is encoded by one or two bytes. The encoding is based on the particular code page number. Many COM Automation applications store text data in Unicode encoding (Office, Databases etc.). Unicode text does not rely on the current code page setting, since every character is encoded by two bytes. The previous versions of VbScriptXtra used system's default ANSI code page to convert Unicode data into MBCS required by Director. Now you may specify particular code page number to be used in text conversion routines of the xtra. See `CodePage` wrapper's property for more details. By default the xtra behaves the same way as it did before.

March 24th, 2002

Version 1.02.003 Build 20
Fixed a bug, which prevented events from exe servers (Word, Excel) to be passed to the events handler. Events called by in-proc servers (ADO) worked properly.

Fixed a bug, which caused Director to crash on quitting if events handler was used.

Events handling code was improved to allow VbScriptXtra to accept events with named arguments (Excel uses named arguments). Arguments list that is passed to events handler parent script now changed to a property list. Property names in that list directly correspond to names of arguments. See description of the `EventsHandler` property for more details.

Applied a workaround to prevent quitting Director while calling 'Quit' event (Word uses it) to a events handler parent script. 'Quit' event is send now as '_Quit' with leading underscore.

Events handler parent script gets every event twice now. The first one is sent in a direct form:

```
on eventName me, argsList
```

Then the universal handler is called:
`on IncomingEvent me, EventSymbol, argsList`

This improvement helps discovering what is sent by a COM Automation object. Note: `IncomingEvents` gets argsList modified by the first handler (if any).

January 11th, 2002 **Version 1.02.002 Build 18**

Xtra's code was optimized a bit. Optimization involves a 'GetProperty' method of the wrapper instance. This method first checks for enumeration property values to handle cases like `put rst.adUseServer` and then if property is not found it is passed to the wrapped automation object. Search of those optimization values was simple linear scan of array, which happens with every property getting.

Now, linear search is replaced to binary search, which is significantly faster when there are a lot of enumeration values. Performance of 'GetProperty' operation like `attr=cnn.Attributes` has increased up to ten percents.

December 13th, 2001 **Version 1.02.001**

Added a detection of the 'busy' state of the server application. The problem arose for example with Excel being in a cell editing state. VbScriptXtra failed with Lingo error 'Property or method not found' when accessing any method or property of Excel while it was in 'busy' state.

Now VbScriptXtra waits for 5 seconds and then if the application is still busy invokes standard OLE dialog which informs a user about 'busy' state of an application and suggests to either cancel the call or switch to that application and solve the problem or just retry. Simple (non-cascading) property access does not generate now a 'Property not found' Lingo alert when server application is in busy state. Cascading properties still may issue such error.

November 18th, 2001 **Version 1.02.000**

Added support for `#True` and `#False` symbols to VbScriptXtra typecasting routines. Now it is possible to get Boolean COM Automation values from symbols `#True` and `#False`. See typecasting for more details.

November 11th, 2001 **Version 1.01.000**

Added support for COM Automation Events. VbScriptXtra wrapper objects now support `EventsHandler` property. Events generated by wrapped automation objects are passed to a parent script attached to this property of the wrapper.

November 4th, 2001 **Version 1.00.005**

Updated automatic type casting rules when converting Lingo linear or property list into COM SafeArray. Now VbScriptXtra scans all list entries and sees whether they are of unique type (float or integer). If so it creates a SafeArray of that type (double or integer respectively). Otherwise it creates a SafeArray of Variants.

October 29th, 2001 **Version 1.00.004**

Fixed a bug with Automation objects reference assignment. It always used PropertyPut even when it should be PropertyPutRef. It is possible that some application might behave incorrectly with it, although I did not noticed anything caused by this bug. Fixed.

Fixed a minor bug with outputting error messages into Director's Messages window. Backslashes disappeared from message text. Fixed.

Added 'return' function handling, to allow Director to actually return a Lingo wrapper for ADODB.Connection object. This connection object may accept any function name, since it tries to execute a stored procedure in this way. So using

```
return cnn
```

did not worked as expected. Fixed.

Added 'underscore handling'. Now VbScriptXtra removes a single underscore from the beginning of any symbol passed to the wrapper as method's or property's name (if any). This allows eliminating a Director's problem of not passing certain names to the xtra at all. Currently noticed names are 'Delete' and 'Append'. Attempt to invoke Delete method of some Automation object generates a Lingo error before Delete is passed to the xtra (D7, D8, D8.5). The same way behaves D8.5 with Append method. To eliminate this problem use: `object._Delete()` `object._Append()` VbScriptXtra will remove the first underscore before passing method name to the wrapped Automation object.

October 15th, 2001 **Version 1.00.003**
Added support for BinaryXtra.

October 12th, 2001 **Version 1.00.002**
Bug fix: `Failed` and `Succeeded` properties always indicated that everything was good, even if `LastError` property contained some error description. The internal last error flag was incorrectly cleared before setting or getting any property including `Failed` and `Succeeded`. Now the internal last error flag is cleared right before accessing properties or methods of a wrapped object.

June 26th, 2001 **Version 1.00.001**
The first public release.

Technical details

Information about technical details of VbScriptXtra, supported and unsupported features of COM Automation technology, implementation details concerning Lingo syntax.

Default object's property

VbScriptXtra does not support default object's property due to differences between Visual Basic syntax and Lingo syntax. Visual Basic uses different assignment operators for assigning reference to the object and assigning value of other data type. Different assignment operators allow VB interpreter to distinguish between using object reference and using the default property of that object. Lingo does not allow differentiating these two cases, therefore you have to specify all property names in most cases.

Default object's method (or indexed property)

Default object's method (indexed property) is partially supported by VbScriptXtra. Usually, any collection object provides default method `Item(index)` which allows access to the item of a collection by default. For example, following lines are the same:

```
put objWord.Documents.Item(1).Name
put objWord.Documents(1).Name
```

Using `objWord.Documents(1)` actually calls `objWord.Documents.Item(1)`. But you cannot use default methods on the top level of cascading properties. The following example will generate Lingo error:

```
objDocs=objWord.Documents
put objDocs(1).Name -- Lingo error here
```

This sample will not work in Lingo, since Lingo thinks `objDocs` is the global handler name and it will try to execute it and will probably fail.

Events

VbScriptXtra supports COM Automation Events since version 1.01.000. VbScriptXtra wrapper objects provide an `EventsHandler` property. Events generated by wrapped automation objects are passed to a parent script attached to this property of the wrapper.

Here is the sample of the universal events handler parent script, which can be used with any COM Automation object that fires events:

```
on new me
  return me
end

on IncomingEvent me, event, args
  put event,args
end
```

Simply assign the instance of this parent script to an `EventsHandler` property of a wrapper of required COM Automation object:

```
object.EventsHandler=new(script"TheNameOfParentScript")
```

and see which events it fires.

Named method's arguments

VbScriptXtra does not support named arguments since they are not supported by Lingo. In Visual Basic you may use following syntax:

```
obj.Method paramName:=actualValue
```

VbScriptXtra does not provide this feature.

Optional and missing method's arguments

Optional and missing arguments are supported by VbScriptXtra, but Lingo requires you to use `VOID` value to indicate missing argument in the middle of the parameters list. Missing arguments in the end of the argument list may be safely skipped. Default values will be used by automation object.

Passing parameters by reference

VbScriptXtra supports arguments passed by reference, although it requires some special conventions. Lingo passes simple type values by value, but automation objects sometimes rely on arguments passed by reference. VbScriptXtra wrapper accepts parent script instances as method arguments. Once wrapper encounters such argument it will use its 'Value' property as an actual argument of the automation object's method. Then after method is executed wrapper will put the updated argument value to 'value' property of the parent script instance. So, if you expecting modified argument value you will have to create a simple parent script instance, set its value property with actual argument value, use that instance as an argument to wrapper object's method and then get the updated value from that instance. Most automation objects' method do not use arguments passed by reference, but sometimes, there is no other way.

Arrays as method's arguments

The current version of VbScript supports only simple arrays as method arguments. VbScriptXtra wrapper will convert any Lingo list to simple array (vector). If list contains a sub list, the conversion will fail and wrapper will set the `lastError` flag with appropriate error description.

Arrays as a return value

VbScriptXtra fully supports arrays returned as result value of any automation call. VbScriptXtra will convert arrays of any depth into list of lists of lists... corresponding depth.

Lingo syntax issues

Lingo syntax is completely different from Visual Basic syntax. These differences lead to minor problems you would better know about.

Director 7 Lingo cascading methods and properties Director 7 has a bug in Lingo interpreter, which requires placing extra brackets to access properties of an object returned by some method. For example,

```
put objAuto.someMethod().someProp -- Lingo error here
```

will generate Lingo error. To avoid it you have to bracket method call:

```
put (objAuto.someMethod()).someProp -- Ok
```

Director 8 and 8.5 does not have this problem.

Using square brackets

When calling method with single argument or accessing indexed property with single index, it is possible to use either normal or square brackets. For example following Lingo syntax is possible with VbScriptXtra wrappers:

```
put rst.fields["FiledName"].Value -- works in D7 too
put rst.fields("FiledName").Value -- does not work in D7, see above
```

Using wrapper instance as first argument of a method

Take care while passing VbScriptXtra wrapper instance as the first argument to a movie handler. This may cause problems with some automation objects. The problem arises from Lingo supporting both original and dot syntax. When Lingo interpreter encounters a method call, it checks whether its first argument is an object instance. So it tries to invoke a method of that object with the same name. If this call fails Lingo searches for a movie handler with this name and calls it if successful.

VbScriptXtra wrapper instance accepts any method name and tries to pass it to the wrapped automation object. Most of automation objects support a fixed set of methods, so the wrapper is capable to find out whether wrapped object supports the particular method or not. Such objects do not cause problems and are correctly passed to the movie handler if they do not support the same method.

There is at least one automation object, which behaves differently. It is ADODB.Connection object. Its instances accept any method names (not only supported directly), since Connection object may try to execute the stored database procedure, which may exist in database. If stored procedure exists it is executed, otherwise it generates corresponding error.

This behavior does not allow VbScriptXtra wrapper to know whether such automation object supports particular name or not. So, avoid passing wrapper instances as the first argument of movie handlers, since you may unintentionally call a method of this object instead of your movie handler.

VbScriptXtra overrides several standard Lingo calls

The problem described above causes VbScriptXtra wrapper to override several standard Lingo calls to ensure you will get expected results. For example, simple Lingo operation like `put objAuto` Lingo interpreter processes it in following order. At first, it invokes the method 'put' of objAuto instance. If objAuto is a VbScriptXtra wrapper instance, it searches method 'put' among methods supported by the wrapped automation object. In most cases it will not find it and then Lingo will execute its own 'put' implementation. But, what happens if object supports this method or automation object supports any methods like ADODB.Connection? Probably, you will get unexpected results.

To eliminate this problem, VbScriptXtra wrapper has its own implementation of the following Lingo commands:
`put objAuto` - puts a brief information about wrapper and wrapped object,
`voidP(objAuto)` - returns false,
`integerP(objAuto)` - returns false,
`stringP(objAuto)` - returns false,
`objectP(objAuto)` - returns true,
`return objAuto` - performs a Lingo return command.

So, these methods are processed by the VbScriptXtra wrapper itself, they are not ever passed to the wrapped automation object. From the other side, if Automation object contains one of overridden methods, you are still able to call it using 'underscore handling'. See below.

'Underscore handling'

VbScriptXtra removes a single underscore (if any) from the beginning of any symbol passed to the wrapper object as a method or a property name. This allows to eliminate a Director's problem of not passing certain names to the xtra at all.

Currently noticed names are 'Delete' and 'Append'. Attempt to invoke Delete method of the wrapped Automation object generates a Lingo error before Delete is even passed to the xtra (D7, D8, D8.5). The same way behaves D8.5 with Append method. To eliminate this problem use:

```
object._Delete()
object._Append()
```

VbScriptXtra will remove the first underscore before passing method name to the wrapped Automation object.

Xtra-level functions, provided by VbScriptXtra:

`Init(bDebugMode)` - Initializes VbScriptXtra and optionally sets debugging mode for newly created wrapper instances.

`CreateObject(strProgId)` - Creates specified automation object and VbScriptXtra wrapper instance to allow Lingo access to the newly created automation object.

`GetObject(strProgId)` - Gets already existing specified automation object if any and VbScriptXtra wrapper instance to allow Lingo access to this automation object.

`Version()` - Returns the version of VbScriptXtra.

xtra "VbScriptXtra".CreateObject(strProgId)

Syntax

`objAuto=CreateObject(xtra"VbScriptXtra",strProgId)` or
`objAuto=xtra("VbScriptXtra").CreateObject(strProgId)`

Parameters

`strProgId` - a string specifying which object to create. For a list of possible ProgIds see FAQ section.

Returns

`Object`: if successful, returns new VbScriptXtra wrapper instance for newly created automation object `ProgId`.

`String`: if failed, returns string with error description.

Description

This method is an analogue to Visual Basic's CreateObject. It is used to create new automation objects. Therefore, it is a main entry point while using VbScriptXtra.

Every creatable automation object defines its own progId. For example, use "Word.Application" to run Microsoft Word.

Sample

```
on navigate
-- Creating a new instance of Internet Explorer
ie=CreateObject(xtra"VbScriptXtra","InternetExplorer.Application")
-- Navigate to URL
ie.navigate("www.XtraMania.com")
-- ie is hidden now, making it fullscreen
ie.fullscreen=true
-- hiding extra interface elements
ie.menubar=false
ie.toolbar=false
ie.statusbar=false
```

```
-- Waiting while page is complete
repeat while ie.busy
put "Waiting for IE"
end repeat

-- Here we are, ready and fullscreen!
ie.visible=true
put "Ops!"
end
```

xtra "VbScriptXtra".GetObject(strProgId)

Syntax `objAuto=GetObject(xtra"VbScriptXtra",strProgId)` or `objAuto=xtra("VbScriptXtra").GetObject(strProgId)`

Parameters `strProgId` - a string specifying which object to create. For a list of possible ProgIds see FAQ section.

Returns `Object`: if successful, returns new VbScriptXtra wrapper instance for already existing automation object `ProgId`.

`String`: if failed, returns string with error description.

Description `GetObject` is used to get access to already existing automation objects. For example, to get access to the running Microsoft Word use:
`objWord=xtra"VbScriptXtra".GetObject("Word.Application")`

xtra "VbScriptXtra".Init(bDebugMode)

Syntax `bSuccess=Init(xtra"VbScriptXtra",bDebugMode)` or `bSuccess=xtra("VbScriptXtra").Init(bDebugMode)`

Parameters `bDebugMode` - a boolean value specifying whether to set debug mode for newly created wrapper instances. Note: the value of this param affects wrapper instances created by further calls to `CreateObject` or `GetObject` methods. When debug mode is set, wrapper instance outputs all information about errors in Messages window.

Returns `true` if successful, `false` otherwise.

Description Initializes VbScriptXtra and optionally sets debugging mode for newly created wrapper instances. Debugging mode is highly recommended while investigation what VbScriptXtra can do, since it will output all error description information right in Messages window. There is no need to set debug mode in release versions. You may safely call Init several times to enable or disable debug mode default setting.

`xtra "VbScriptXtra".Version()`

Syntax `strVersion=Version(xtra"VbScriptXtra")` or
`strVersion=xtra("VbScriptXtra").Version()`

Returns `String` with version info in a form: `"VbScriptXtra.1.00.001"`

Description Returns the version of VbScriptXtra

Properties and methods provided by VbScriptXtra automation object wrapper

Automation Object Wrapper is a key component of VbScriptXtra. It is used to pass your Lingo method and property calls onto wrapped automation object and to return the corresponding result, using automatic type casting and error handling mechanism.

Error handling properties:

- `objAuto.DebugMode` - gets or sets debugging mode of the wrapper instance.
- `objAuto.Succeeded` - returns true if previous wrapper access to the automation object has succeeded.
- `objAuto.Failed` - returns true if previous wrapper access to the automation object has failed.
- `objAuto.LastError` - returns the last error description if any. Error description usually is provided by Automation Object or COM library.

Autodocumentation feature:

- `interface(objAuto)` - returns a string with Automation Object interface description.
- `enums(objAuto)` - returns a string with Automation Object enumerations description.

COM Automation Events support:

- `objAuto.EventsHandler` - gets or sets a parent script instance used to handle events generated by wrapped automation object.

Advanced Unicode text support:

- `objAuto.CodePage` - gets or sets a code page number to be used in text conversion from Unicode to Director's MBCS and vice versa.

Overrided standard Lingo commands:

- `put objAuto` - puts a brief information about wrapper and wrapped object.
- `voidP(objAuto)` - returns false. `integerP(objAuto)` - returns false.
- `stringP(objAuto)` - returns false. `objectP(objAuto)` - returns true.
- `return objAuto` - Lingo 'return' command is executed

Any other properties and methods are passed to the wrapped automation object. Wrapper instance trims the first underscore '_' character (if any) from any method or property name

passed to the wrapped Automation object. The actual implementation of a method or a property depends on particular automation object. VbScriptXtra wrapper provides necessary type casting and error handling mechanism.

`objAuto.CodePage`

Syntax `nCodePage=objAuto.CodePage` `objAuto.CodePage=nCodePage`

Gets Integer value, which indicates the code page number used in Unicode/MBCS text conversion routines.

Sets Integer value, which indicates the code page number to be used in Unicode/MBCS text convertion routines.

Description The `CodePage` property allows the xtra to be used in multilingual environment. Director uses MBCS (Multibyte Character Set). Every character is encoded by one or two bytes. The encoding is based on the particular code page number. Many COM Automation applications store text data in Unicode encoding (Office, Databases etc.). Unicode text does not rely on the current code page setting, since every character is encoded by two bytes. This property defines particular code page number to be used in text conversion routines of the xtra.

By default, the `CodePage` property is 0 - ANSI code page. It defines the default behavior for the system.

The `CodePage` property affects all text conversion operations initiated by this instance of the wrapper. All wrappers created by this wrapper inherit the value of `CodePage` property. In other words, all VbScriptXtra wrappers created by xtra-level method `CreateObject` get the default value of the code page, which is zero. Wrapper instances derived from some wrapper instance inherit the code page setting.

Take care when changing the default value of this property, since inappropriate code page number may result in empty string as a result of text conversion. Below is the list of possible code page numbers:

Code page number	Meaning
0	ANSI code page

2	Macintosh code page
1	OEM code page
42	Symbol code page (Win2k)
3	The current thread's ANSI code page (Win2k)
65000	Translate using UTF-7 (Win2k, NT 4.0)
65001	Translate using UTF-8 (Win2k, NT 4.0)
037	EBCDIC
437	MS-DOS United States
500	EBCDIC "500V1"
708	Arabic (ASMO 708)
709	Arabic (ASMO 449+, BCON V4)
710	Arabic (Transparent Arabic)
720	Arabic (Transparent ASMO)
737	Greek (formerly 437G)
775	Baltic
850	MS-DOS Multilingual (Latin I)
852	MS-DOS Slavic (Latin II)
855	IBM Cyrillic (primarily Russian)
857	IBM Turkish
860	MS-DOS Portuguese
861	MS-DOS Icelandic
862	Hebrew
863	MS-DOS Canadian-French
864	Arabic

865	MS-DOS Nordic
866	MS-DOS Russian
869	IBM Modern Greek
874	Thai
875	EBCDIC
932	Japanese
936	Chinese (PRC, Singapore)
949	Korean
950	Chinese (Taiwan; Hong Kong SAR, PRC)
1026	EBCDIC
1200	Unicode (BMP of ISO 10646)
1250	Windows 3.1 Eastern European
1251	Windows 3.1 Cyrillic
1252	Windows 3.1 US (ANSI)
1253	Windows 3.1 Greek
1254	Windows 3.1 Turkish
1255	Hebrew
1256	Arabic
1257	Baltic
1361	Korean (Johab)
10000	Macintosh Roman
10001	Macintosh Japanese
10006	Macintosh Greek I
10007	Macintosh Cyrillic

10029	Macintosh Latin 2
10079	Macintosh Icelandic
10081	Macintosh Turkish

`objAuto.DebugMode`

Syntax
```
bDebugMode=objAuto.DebugMode
objAuto.DebugMode=bDebugMode
```

Gets Boolean value, which indicates whether wrapper currently in debug mode.

Sets Boolean value. Use true to set debug mode. Use the false value to clear the debug mode of the wrapper instance.

Description VbScriptXtra automation object wrapper supports special debugging mode. While the debug mode is set wrapper instance outputs any error messages directly to the Message window every time error happens. By default, VbScriptXtra wrappers created by `CreateObject` call inherit the xtra's default value for debugMode. You may change this default with `Init` xtra-level method. Wrappers created by other wrappers inherit this setting.

Note: While you are just investigating automation capabilities it is better to set Debug Mode by default, to ensure you always know if something goes wrong. It might be important while using cascading properties, since every property returning Automation Object will be wrapped by a new instance of VbScriptXtra wrapper. So, you may skip useful error description.

`objAuto.EventsHandler`

Syntax
```
objHandler=objAuto.EventsHandler
objAuto.EventsHandler=objHandler
```

Gets `VOID` if events are not handled.

`Instance` of parent script, which handles events generated by automation object.

Sets

`VOID` to stop handling events.

`Instance` of parent script, which will handle events generated by automation object.

Description

Use `EventsHandler` property to set a handler for Automation events generated by the wrapped Automation object.

Use ObjectBrowserXtra to see which Automation objects support events. Any Automation object may provide an outgoing interface describing which events it can fire.

After `EventsHandler` property is set to a valid parent script instance, wrapper starts listening for events. Once a particular event occurs, wrapper tries to call a corresponding method of the attached parent script instance. Event parameters are passed as Lingo property list. Using Lingo list allows event handler to return values via parameters passed by reference.

The code for event handler parent script may look like:

```
-- Sample EventHandler parent script
on new me
  return me
end

on SomeEventName me, argsList
  arg1=argsList[#ArgName1]  arg2=argsList[#ArgName2]
  put "SomeEventName fired with arg1="&arg1&&"arg2 =
  "&arg2  argsList[2]=SomeNewValue
end

on IncomingEvent me, event, args
  put event,args
end
```

Events handler parent script gets every event twice. The first event is sent in a direct form:

```
on eventName me, argsList
```

Then the universal handler is called:

```
on IncomingEvent me, EventSymbol, argsList
```

This improvement helps discovering what is sent by a COM Automation object. Note: `IncomingEvents` gets argsList modified by the first handler (if any).

Sample

Sample below demonstrates using events with ADODB.Connection object:

```
--****************************************************
-- Here is the code for EventHandler parent script
on new me
  return me
end

on IncomingEvent me, event, args
  put event,args
end

on ConnectComplete me, args
  pError=args[1]
  adStatus=args[2]
  pConnection=args[3]

  put "ConnectComplete"
  if(adStatus<>1) then alert pError.Description
end

on Disconnect me, args
  adStatus=args[1]
  pConnection=args[2]

  put "Disconnect"
end

on ExecuteComplete me, args
  RecordsAffected=args[1]
  pError=args[2]
  adStatus=args[3]
  pCommand=args[4]
  pRecordset=args[5]
  pConnection=args[6]

  put "ExecuteComplete"
end

on InfoMessage me, args
  pError=args[1]
  adStatus=args[2]
  pConnection=args[3]

  put "InfoMessage"
```

```
end

on WillConnect me, args
  ConnectionString=args[1]
  UserID=args[2]
  Password=args[3]
  Options=args[4]
  adStatus=args[5]
  pConnection=args[6]

  put "WillConnect"

  -- Creating new connection string
  cnnStr="Provider=Microsoft.Jet.OLEDB.4.0;"
  -- Microsoft Jet provider for MS Access databases
  cnnStr=cnnStr&"Data Source=D:\Temp\TestDB.mdb;"
  cnnStr=cnnStr&"Mode=Read|Write;"

  -- return it to the connection object
  -- via referenced parameter
  args[1]=cnnStr
end

on WillExecute me, args
  Source=args[1]
  CursorType=args[2]
  LockType=args[3]
  Options=args[4]
  adStatus=args[5]
  pCommand=args[6]
  pRecordset=args[7]
  pConnection=args[8]

  put "WillExecute"
end

-- End of the code for EventHandler parent script
-- *****************************************************
```

Name this script as "ConnectionEvents". Then try to execute following lines right in Director's messages window:

```
-- Setting debug mode to true
xtra("VbScriptXtra").Init(true)

-- Creating an instance of the ADODB.Connection object
cnn=createObject(xtra"VbScriptXtra",
"ADODB.Connection")

-- Creating an instance of
-- the events handler parent script
evnts=new(script"ConnectionEvents")
```

```
-- Attaching handler to a wrapper
cnn.EventsHandler=evnts

-- Opening connection without explicitly
-- specifying connection params
-- Connection string should be set by
-- the events handler
cnn.Open()
cnn.Close()
```

To get more information about ADO events you may use ObjectBrowserXtra or see ADO documentation.

`objAuto.Failed`

Syntax `bResult=objAuto.Failed`

Returns Boolean value indicating whether last attempt to access wrapped automation object has failed.

Description Use `Failed` property to check whether error occurred. Be careful with cascading properties, since cascaded access is usually implemented using temporary wrapper instances. So, if error is encountered deeper then at the first level, you may not get a possibility to know that unless Lingo error will be produced. Use debug mode to track down such conditions. Note also, that misspelled properties and methods will produce corresponding Lingo error in most cases.

See Debugging for related information.

`objAuto.LastError`

Syntax `strErrorDescription=objAuto.LastError`

Returns String with last error's description or empty string if the last call was successful.

Description Use `LastError` property to get the description of error occurred. Use

debug mode to automatically get error descriptions in Messages window.

See Debugging for related information.

Sample

```
if objAuto.Failed then
  put objAuto.LastError
end if
```

objAuto.Succeeded

Syntax `bResult=objAuto.Succeeded`

Returns Boolean value indicating whether last attempt to access wrapped automation object has completed successfully.

Description Use `Succeeded` property to check whether last operation with automation object completed successfully. Be careful with cascading properties, since cascaded access usually implemented using temporary wrapper instances. So, if error is encountered deeper then at the first level, you may not get a possibility to know that, unless Lingo error will be produced. Use debug mode to track down such conditions. Note also, that misspelled properties and methods will produce corresponding Lingo error in most cases.

See Debugging for related information.

enums(objAuto)

Syntax `strEnums=Enums(objAuto)` or `strEnums=objAuto.Enums()`

Returns String value with the list of available enumerations for the wrapped automation object.

Description Use `enums()` method to invoke autodocumentation feature of VbScriptXtra. It will show you which enumeration constants you can use with particular automation object.

See Autodocumentation feature for more information.

Sample

```
objWord=xtra"VbScriptXtra".CreateObject("Word.
  Application")
put enums(objWord)
```

`interface(objAuto)`

Syntax

`strInterface=interface(objAuto)` or
`strInterface=objAuto.interface()`

Returns

String value with the interface description of the wrapped automation object.

Description

Use `interface()` method to invoke autodocumentation feature of VbScriptXtra. It will show you what you can do with particular automation object.

See Autodocumentation feature for more information.

Sample

```
objWord=xtra"VbScriptXtra".CreateObject("Word.
  Application")
put interface(objWord)
```

VbScriptXtra automation object wrapper - automatic type casting

VbScriptXtra performs automatic type casting to correctly transfer data between Lingo and Automation object and vice versa. VbScriptXtra implicitly performs typecasting operations during processing Lingo method arguments, returning values, and property values. Automation and Visual Basic supports rather large amount of data types. Lingo has its own Director specific data types. So, VbScriptXtra may not find suitable conversion in all cases, although it provides conversion in the most cases. If VbScriptXtra does not know how to convert the value it will report an error. See Mapping Lingo types to COM Automation types and Mapping COM Automation types to Lingo types topics below for further details.

Mapping Lingo types to COM Automation types

This conversion is taking place when VbScriptXtra wrapper passes any arguments to the wrapped automation object. This includes assigning property values of the wrapped automation object. The table below describes Lingo types that are recognized by VbScriptXtra wrapper and into which types they are converted.

Lingo type	Lingo Value (if any)	Automation type
Symbol	#Null	NULL
Symbol	#True	Boolean (true)
Symbol	#False	Boolean (false)
Symbol	other symbols	String
Integer		signed integer 4 bytes
Float		Float 8 bytes
String		String
Date		Float
List of floats		SafeArray (Vector) of Doubles
List of integers		SafeArray (Vector) of Integers
Property or Linear List		SafeArray (Vector) of Variants
VOID	VOID	Missing value
Parent Script Instance		Depends on 'Value' property

VbScriptXtra wrapper		Automation object
BinaryXtra wrapper		SafeArray (Vector) of Bytes

Director String values are MBCS text. It is converted to Unicode by the xtra. Text conversion routine uses `CodePage` setting of the wrapper instance that requested the conversion. The code page number directly affects the result of text conversion. Take care when changing the default value of this property, since inappropriate code page number may result in empty string as a result of conversion.

The Symbol type is native to Director, COM knows nothing about it. The actual integer value of any symbol is guaranteed to be the same only during current Director session. So, typecasting routine of VbScriptXtra converts Lingo symbol value into the corresponding string, except cases listed in the table. So value `#SomeSymbol` will be converted to string `"SomeSymbol"`.

The symbol value `#Null` is treated as COM Automation `Null` value. It is often used in databases and in VB to mark empty references to objects.

Symbols `#True` and `#False` are converted into corresponding Boolean values. Note the difference between symbol `#True` and Lingo constant value `true`. In Lingo value `true` is the same as Integer value `1`, so it is converted as Integer value by VbScriptXtra. Sometimes, it may be important to pass exactly Boolean value to the Automation object. That is why special processing of symbols `#True` and `#False` was added to the VbScriptXtra since version 1.02.000.

Special note about Date values. The Date/Time value in COM Automation is actually represented as a float number where the whole part is the number of days since the 1st of January 1901, and the fraction part represents the time. Director uses its own Date/Time values. VbScriptXtra provides a conversion of these values into COM date/time values, but due to a bug in Director 8, the conversion gets the wrong results if the date value is outside this range: date(1901,12,14) to date(2038,1,19). Conversion of date values inside this range works correctly. Director 7.02 and Director 8.5 work correctly over all time period available to COM Automation.

Both linear lists and property lists are converted into SafeArray values. The current VbScriptXtra version supports conversion only one dimension arrays. So conversion will fail if Lingo list contains sub list. Inverse conversion works without this limitation. Note: SafeArrays may differ by type of its element. Since version 1.00.005r01 VbScriptXtra scans the Lingo list and see whether list entries all have unique data type (either integer or float). If VbScriptXtra detects that all list entries are of unique type it creates a SafeArray of that type. Otherwise it creates a SafeArray of Variants. Note: VbScriptXtra creates a SafeArray with lower bound set to zero.

VOID Lingo values are usually treated as missing argument, therefore it is converted into corresponding COM Automation value which indicates missing argument.

Other VbScriptXtra wrapper as an argument is converted to the corresponding wrapped automation object reference. So you can safely use VbScriptXtra wrappers with corresponding automation objects where other objects expect them.

VbScriptXtra supports arguments passed by reference through parent script instances. Once wrapper encounters such argument it will use its 'Value' property as an actual argument of the automation object's method. Then after method is executed wrapper will put the updated argument value to 'value' property of the parent script instance. So, if you expecting modified argument value you will have to create a simple parent script instance, set its value property with actual argument value, use that instance as an argument to wrapper object's method and then get the updated value from that instance.

VbScriptXtra can handle large binary data contained in BinaryXtra wrapper. If such wrapper is passed as an argument, its data is converted to the SafeArray of unsigned chars. For more information about BinaryXtra see its documentation.

Mapping COM Automation types to Lingo types

This conversion is taking place when VbScriptXtra wrapper returns any value returned by the wrapped automation object and when updating arguments passed by reference. This includes getting property values of the wrapped automation object. The table below describes COM Automation types, which are recognized by VbScriptXtra wrapper and into which Lingo types they are converted.

Automation type	**Lingo type**	**Value (if any)**
EMPTY	`VOID`	VOID
NULL	`Symbol`	#Null
Integer (signed/unsigned), 1,2,4 bytes	`Integer`	
Float 4,8 bytes	`Float`	
Numeric	`Float`	
Date	`Float`	
String	`String`	
Boolean	`Integer`	1 or 0
Currency	`Float`	
GUID	`String`	
SafeArray of Variants	Linear `List`	

Automation object	VbScriptXtra wrapper	
SafeArray of Bytes (BLOB, OLE, Image)	BinaryXtra wrapper	

Director String values are MBCS text. COM Automation object uses Unicode as a text encoding. Unicode text is converted to MBCS by the xtra. Text conversion routine uses `CodePage` setting of the wrapper instance that requested the conversion. The code page number directly affects the result of text conversion. Take care when changing the default value of this property, since inappropriate code page number may result in empty string as a result of conversion.

Empty COM Automation values are converted to the VOID Lingo values.

Null value, which is often used in databases and in VB to mark empty references to objects, is converted to Lingo symbol #Null.

Any integer values are converted to 4 byte signed Integer value native to Director.

Any float or numeric or currency values are converted to 8-byte float value native to Director.

Boolean value in COM Automation usually represented as -1 for true and 0 for false. It is converted to integer values 1 and 0 accordingly.

GUID values are converted in string representation and then are converted into Lingo string values.

SafeArrays of Variants of any depth are converted to the lists of lists of lists... with the corresponding depth.

If Automation object returns a reference to another Automation object, the new VbScriptXtra wrapper is created and initialized with this reference. The new wrapper is returned to Lingo.

Date values are converted to the corresponding float values. Advanced date/time conversion routines will be available soon.

SafeArrays of Bytes (BLOB, OLE or Image database fields) are converted to the BinaryXtra wrappers containing those binary data. This conversion works if BinaryXtra is detected, otherwise 'Cannot convert COM value to Lingo value' error is reported. BinaryXtra is free. It is available in Downloads section. For more information about BinaryXtra see its documentation.

VbScriptXtra Samples

Excel - sample script runs Excel application (if any) creates new workbook and puts a simple message into the first cell of the first sheet.

Internet Explorer - sample script silently navigates to an URL, waits while downloading is ready and then shows an IE window in specified location.

Word - simple script runs Word application (if any) creates new document and types a simple message at the top of the page.

ADO - simple script for opening ADO recordset for reading or writing to Microsoft Access database. Demo movie "VbScriptXtra & ADO databasing guided tour" is available for download.

MSNMessenger - simple script that retrieves contacts list from MSN Messenger.

Open ADO Recordset sample script

```
-- Handler creates a new recordset object
-- connects it to the MS Access database dbPath
-- sets access rights to read or read/write
-- depending on bReadWrite parameter
-- executes SQL query and
-- returns resulting recordset object if successful or
-- string with error description otherwise


on OpenRecordset dbPath, sql, bReadWrite
  if voidP(bReadWrite) then bReadWrite=false
  if voidP(sql) then return "OpenRecordset: Required parameter
    is missing: "&sql
  if voidP(dbPath) then return "OpenRecordset: Required parameter
    is missing: "&dbPath

  -- Creating recordset object
  rst=createObject(xtra "VbScriptXtra","ADODB.Recordset")
  if not objectP(rst) then return rst

  -- Building connection string
  cnnStr="Provider=Microsoft.Jet.OLEDB.4.0;"
  -- Microsoft Jet provider for MS Access databases
  cnnStr=cnnStr&"Data Source="&dbPath&";"

  if bReadWrite then
    cnnStr=cnnStr&"Mode=Read|Write;"
  else
    cnnStr=cnnStr&"Mode=Read;"
  end if

  rst.ActiveConnection=cnnStr
  if rst.failed then return rst.lastError

  if bReadWrite then
    rst.lockType=rst.adLockPessimistic
    rst.CursorType=rst.adOpenKeyset
  else
    rst.lockType=rst.adLockReadOnly
    rst.CursorType=rst.adOpenStatic
  end if

  rst.Open(sql)
  if rst.failed then return rst.lastError

  return rst
end
```

Where to find more info about ADO?

At first, ADO is fully documented at msdn.microsoft.com.

VbScriptXtra provides autodocumentation feature allowing you to see what you can do with particular object instance. Just create a recordset object and type in Messages window:

```
put interface(rst)
put interface(rst.fields)
put interface(rst.fields(0))
```

You may also use ObjectBrowser xtra - autodocumentation companion for VbScriptXtra - to view all available ADO interfaces, methods, properties and enumerations.

```
xtra("ObjectBrowser").Browse(rst)
xtra("ObjectBrowser").Browse(rst.fields)
xtra("ObjectBrowser").Browse(rst.fields(0))
```

It is available at Downloads for free.

"Hello World!" for Excel = "Hello Excel!"

This handler will run Excel application (if any) create new workbook and print simple message in the first cell of the first sheet.

```
on testExcel
  -- Creating a new instance of Microsoft Excel
  excel=xtra("VbScriptXtra").CreateObject("Excel.Application")

  -- Checking whether object is created
  if not objectP(excel) then
    alert "Failed to create Excel.Application:"&RETURN&excel
    exit
  end if

  -- Setting application's window params
  excel.visible=true
  excel.DisplayFullScreen=false
  excel.left=100
  excel.top=100
  excel.width=400
  excel.height=300

  -- Creating new workbook
  doc=excel.workbooks.add()

  -- Getting access to the first work sheet of
  -- the newly created work book
  sheet=doc.worksheets(1)

  -- Arranging work book window
  excel.windows.arrange()

  -- Setting the value of the left-top cell
  -- Extra parentheses required for Director 7
  (sheet.cells(1,1)).Value="Hello Excel!"

  -- Setting the font style of the left-top cell
  (sheet.cells(1,1)).Style.Font.Bold=true

  -- Setting the width and height of the cell using range property
  -- Square brackets may be used with single argument
  sheet.range["A1:A1"].rowHeight=64
  sheet.range["A1:A1"].columnWidth=16
end
```

Where to find more info about Excel?

At first, Microsoft Excel contains rather large help system, which contains just everything about Excel. Look at Visual Basic for Excel topics, which describes Excel data model,

objects, methods and properties you can use. Make sure you have installed this part of the help system, since it may not be included in typical installation.

VbScriptXtra provides autodocumentation feature allowing you to see what you can do with particular object instance. Just make variables in the script above global and type in Messages window:

```
put interface(excel)
put interface(excel.WorkBooks)
put interface(sheet)
put interface(sheet.cells(1,1))
```

You may also use ObjectBrowser xtra - autodocumentation companion for VbScriptXtra - to view all available Excel interfaces, methods, properties and enumerations.

```
xtra("ObjectBrowser").Browse(excel)
xtra("ObjectBrowser").Browse(excel.WorkBooks)
xtra("ObjectBrowser").Browse(sheet.cells(1,1))
```

It is available at Downloads for free.

Silent navigation to url

This handler will run hidden Internet Explorer application, tells him to navigate to URL, waits while IE is ready and then makes it visible.

```
on Navigate url, displayRect
  if voidP(url) then url="www.XtraMania.com"

  -- Creating a new instance of Internet Explorer
  ie=xtra("VbScriptXtra").CreateObject("InternetExplorer.
    Application")

  -- Checking whether object is created
  if not objectP(ie) then
    alert "Failed to create InternetExplorer.Application:"&ie
    exit
  end if

  -- Navigate to URL
  ie.navigate(url)

  -- hiding extra interface elements
  ie.menubar=false
  ie.toolbar=false
  ie.statusbar=false

  -- Adjusting Internet Explorer window rect
  if voidP(displayRect) then
    ie.fullScreen=true
  else
    ie.width=displayRect[3]-displayRect[1]
    ie.height=displayRect[4]-displayRect[2]
    ie.left=displayRect[1]
    ie.top=displayRect[2]
  end if

  -- Waiting while page is completed
  start=the ticks
  seconds=0
  repeat while ie.readyState<ie.READYSTATE_LOADED
    time=the ticks
    if (time-start)/60 seconds then
      seconds=(time-start)/60
      put "Waiting for IE:"&&seconds&&"sec(s)"
    end if
  end repeat

  -- Here we are, ready!
  ie.visible=true
```

```
  put "Ops IE is ready!"
end
```

Where to find more info about Internet Explorer?

VbScriptXtra provides autodocumentation feature allowing you to see what you can do with particular object instance. Just make variables in the script above global and type in Messages window:

```
  put interface(ie)
  put interface(ie.document)
```

You may also use ObjectBrowser xtra - autodocumentation companion for VbScriptXtra - to view all available InternetExplorer interfaces, methods, properties and enumerations.

```
  xtra("ObjectBrowser").Browse(ie)
  xtra("ObjectBrowser").Browse(ie.Document)
```

It is available at Downloads for free.

Retrieving contacts from MSN Messenger

This handler will retrieve your contacts from Messenger and simply put it in Messages window.

```
on testMessenger
  -- Creating the Messenger object
  objMsgrObject = xtra("VbScriptXtra").CreateObject("Messenger.
    MsgrObject")
  if not objectP(objMsgrObject) then
    alert "Failed to create messenger:"&RETURN&objMsgrObject
    exit
  end if

   -- Getting users list
  IMsgrUsers = objMsgrObject.List(0)

  -- Retrieving info about every contact in the list
  repeat with counter=0 to IMsgrUsers.count-1
    put IMsgrUsers.Item(counter).FriendlyName & " = " &
     IMsgrUsers.Item(counter).LogonName
  end repeat
end
```

Sent by: Dave Mee, London

Where to find more info about Messenger?

VbScriptXtra provides autodocumentation feature allowing you to see what you can do with particular object instance. Just make variables in the script above global and type in Messages window:

```
  put interface(objMsgrObject)
  put interface(IMsgrUsers)
```

You may also use ObjectBrowser xtra - autodocumentation companion for VbScriptXtra - to view all available Messenger interfaces, methods, properties and enumerations.

```
  xtra("ObjectBrowser").Browse(objMsgrObject)
  xtra("ObjectBrowser").Browse(IMsgrUsers)
```

It is available at Downloads for free.

"Hello World!" for Word = "Hello Word!"

This handler will run Word application (if any) create new document and type simple message at the top of the page.

```
on testWord
  -- Creating a new instance of Microsoft Word
  wordApp=xtra("VbScriptXtra").CreateObject("Word.Application")

  -- Checking whether object is created
  if not objectP(wordApp) then
    alert "Failed to create Word.Application:"&RETURN&wordApp
    exit
  end if

  -- Setting application's window params
  wordApp.visible=true
  wordApp.left=100
  wordApp.top=100
  wordApp.width=400
  wordApp.height=300

  -- Creating new workbook
  doc=wordApp.documents.add()

  -- Arranging document window
  wordApp.windows.arrange()

  -- Typing simple message at the beginning of the document
  -- Extra parentheses required for Director 7
  (doc.range()).InsertAfter("Hello Word!")

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

Where to find more info about Word?

At first, Microsoft Word contains rather large help system, which contains just everything about Word. Look at Visual Basic for Word topics, which describe Word data model, objects, methods and properties you can use. Make sure you have installed this part of the help system, since it may not be included in typical installation.

VbScriptXtra provides autodocumentation feature allowing you to see what you can do with particular object instance. Just make variables in the script above global and type in Messages window:

```
  put interface(word)
  put interface(word.Documents)
```

```
put interface(doc)
put interface(doc.range())
```

You may also use ObjectBrowser xtra - autodocumentation companion for VbScriptXtra - to view all available Word interfaces, methods, properties and enumerations.

```
xtra("ObjectBrowser").Browse(word)
xtra("ObjectBrowser").Browse(doc.range())
```

It is available at Downloads for free.

Contacts

If you wish to contact xtras' author, you may contact me by e-mail: author@xtramania.com

Best regards, Eugene Shoustrov