array

A set of sequentially indexed elements having the same intrinsic data type. Each element of an array has a unique identifying index number. Changes made to one element of an array don't affect the other elements.

Automation object

An object that is exposed to other applications or programming tools through Automation interfaces.

class

The formal definition of an object. The class acts as the template from which an instance of an object is created at run time. The class defines the properties of the object and the methods used to control the object's behavior.

collection

An object that contains a set of related objects. An object's position in the collection can change whenever a change occurs in the collection; therefore, the position of any specific object in the collection can vary.

expression

A combination of keywords, operators, variables, and constants that yields a string, number, or object. An expression can be used to perform a calculation, manipulate characters, or test data.

identifier

An element of an expression that refers to a constant or variable.

member

An element of a collection, object, or user-defined type.

module

A set of declarations followed by procedures.

numeric expression

Any expression that can be evaluated as a number. Elements of an expression can include any combination of keywords, variables, constants, and operators that result in a number.

object

A combination of code and data that can be treated as a unit, for example, a control, form, or application component. Each object is defined by a class.

procedure

A named sequence of statements executed as a unit. For example, **Function** and **Sub** are types of procedures.

property

A named attribute of an object. Properties define object characteristics such as size, color, and screen location, or the state of an object, such as enabled or disabled.

run-time error

An error that occurs when code is running. A run-time error results when a statement attempts an invalid operation.

scope

Defines the visibility of a variable, procedure, or object. For example, a variable declared as **Public** is visible to all procedures in all modules. Variables declared in a procedure are visible only within the procedure and lose their value between calls.

string expression

Any expression that evaluates to a sequence of contiguous characters. Elements of a string expression can include a function that returns a string, a string literal, a string constant, or a string variable.

Can't execute; script is running

You have attempted to access one or more of the **ScriptControl** object's members while a script is running. Once a script is running, most properties are read-only. Also, methods that affect the members of the **ScriptControl** object can't be accessed while script is running. Examples of these methods include **AddCode** and **AddObject** methods.

Can't set UseSafeSubset property

You can't set the **UseSafeSubset** property. Either the application hosting the **ScriptControl** object has forced it into safe mode or the scripting engine does not support the safety features. Another possibility is that if the scripting engine does support safety features, it may not support changing the **UseSafeSubset** property at all times.Try setting the **UseSafeSubset** property before calling any methods or before setting the **Language** property.

Executing script has timed out

The script has ended because it has taken more time to execute than specified by the **Timeout** property.

Language property not set

You tried to use a method or property that is only allowed when the **Language** property of the **ScriptControl** object has been set to a valid language.

Member is not supported by selected script engine

You tried to access a method or property that the scripting engine specified by the **Language** property doesn't support.

Object is no longer valid

You tried to access an object that is no longer available because **ScriptControl** has been reset.

Error Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scobjErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scobjErrorX":1}
{ewc HLP95EN.DLL,DYNALINK,"Properties":"scobjErrorP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"scobjErrorM"}
{ewc HLP95EN.DLL,DYNALINK,"Events":"scobjErrorE"}

Contains information about compilation and run-time errors that occur.

Remarks

The **ScriptControl Error** object provides a superset of the properties found in the standard Visual Basic **Err** object. The additional properties allow you to provide users with feedback after a detectable error occurs.

Aside from the additional properties, an important difference between the Visual Basic **Err** object and the **ScriptControl Error** object is that the **Error** object is not global, that is, one **Error** object does not handle all **ScriptControl** errors. Each **ScriptControl** has its own **Error** object. In Visual Basic, there is only one **Err** object.

ScriptControl Error object properties are reset to zero or zero-length strings ("") each time the **ScriptControl Language** property is changed or when a call is made to any of the following methods: **Reset**, **AddCode**, **Eval**, or **ExecuteStatement**. Use the **Clear** method to explicitly reset **Error** object properties.

When an error occurs during a call into the **ScriptControl**, the Visual Basic **Err** object properties are set to the same values as those in the **ScriptControl Error** object. When a run-time error occurs within **ScriptControl** code, only the **ScriptControl Error** object properties contain the correction information.

Module Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scobjModuleC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scobjModuleX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"scobjModuleP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"scobjModuleM"} {ewc HLP95EN.DLL,DYNALINK,"Events":"scobjModuleE"}

A **Module** object is the **ScriptControl** member in which all procedure, type, and data declarations are made.

Remarks

What, exactly, goes into a module is determined by the scripting engine in use.

Scripting code, including procedures defined by the scripting engine in use, is added to a **Module** object using the **AddCode** method.

The **ScriptControl** is automatically provided with a **Global** module. Scripting code added to the **ScriptControl** itself always goes into the **Global** module. Scripting code in the **Global** module is available to all other modules in the **ScriptControl**.

Additional modules can be added to the **Modules** collection using the **Add** method. Scripting code added to any additional modules is local to the module in which the code is added.

Modules Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"sccolModulesC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"sccolModulesX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"sccolModulesP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"sccolModulesM"} {ewc HLP95EN.DLL,DYNALINK,"Events":"sccolModulesE"}

The **Modules** collection contains all modules in a **ScriptControl** object, including the **Global** module.

Remarks

The **Modules** collection provides a convenient way to refer to all modules in a **ScriptControl** as a single object.

Individual modules are added using the **Add** method. Specific modules can be returned from the **Modules** collection using the **Item** method, while the entire collection can be iterated using the **For Each...Next** statement.

The **Global** module can always be found using the **GlobalModule** constant as an index to the **Modules** collection.

Procedure Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scobjProcedureC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scobjProcedureX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"scobjProcedureP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"scobjProcedureM"} {ewc HLP95EN.DLL,DYNALINK,"Events":"scobjProcedureE"}

The **Procedure** object is logical unit of code as defined by the scripting engine in use.

Remarks

The **Procedure** object provides entry points to the procedures within a module.

Individual **Procedure** objects are added to the **Procedures** collection of a specific module using the **AddCode** method.

Procedures Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"sccolProceduresC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"sccolProceduresX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"sccolProceduresP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"sccolProceduresM"} {ewc HLP95EN.DLL,DYNALINK,"Events":"sccolProceduresE"}

The **Procedures** collection contains all procedures defined in a specific **Module** object.

Remarks

The **Procedures** collection provides a convenient way to refer to all procedures in a **Module** as a single object.

Individual procedures are added to specified modules using the **AddCode** method. Individual procedures added to the **ScriptControl** are contained in the **Global** module's **Procedures** collection.

Specific procedures can be returned from the **Procedures** collection using the **Item** method, while the entire collection can be iterated using the **For Each...Next** statement.

ScriptControl Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scobjScriptControlC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scobjScriptControlX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"scobjScriptControlP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"scobjScriptControlM"} {ewc HLP95EN.DLL,DYNALINK,"Events":"scobjScriptControlE"}

Enables scripting in an application.

Remarks

The **ScriptControl** provides a simple interface for hosting scripting engines that support ActiveX scripting. The **ScriptControl** object supports the following:

- Any scripting language that supports ActiveX scripting.
- A macro Run dialog box that lists available macros.
- The ability to compile scripts and display rich error information describing any errors that may occur.
- The ability to trap and display run-time errors that occur during script execution.
- The ability to expose object model functionality to scripts.
- The ability to expose global functions to scripts.
- An **Immediate** window.
- The ability to limit a script's execution both in terms of its functionality and its time limitations.
- The ability to use Microsoft script debugging tools to debug scripts written with a scripting-enabled version of Notepad.

AllowUI Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproAllowUIC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproAllowUIX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproAllowUIA"}

Sets or returns a Boolean value that indicates whether a running script, or the **ScriptControl** itself, is allowed to display user-interface elements. Read/write.

Syntax

object.**AllowUI** [= *value*]

The **AllowUI** property has these parts:

Part	Description
object	Required. Can be any object in the "Applies To" list or a reference to it.
value	Optional. Boolean value that is **True** if display of user-interface elements is allowed; **False** if it is not.

Remarks

The **AllowUI** property applies to user-interface elements displayed by the **ScriptControl** itself or user-interface elements displayed by the scripting engines.

In VBScript, for example, if **AllowUI** is **False**, the **MsgBox** statement will not work. Note that if **AllowUI** is **False**, there is no way to notify a user of the occurrence of a TimeOut event

CodeObject Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproCodeObjectC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproCodeObjectX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproCodeObjectA"}

Returns an object that is used to call public members of a specified module. Read-only.

Syntax

object.**CodeObject**

The *object* is the name of a specific module or reference to it.

Remarks

The names of public members of a module are those provided by a programmer using the **ScriptControl** in an application.

Column Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproColumnC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproColumnX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproColumnA"}

Returns a column location indicating the approximate place in the scripting code where an error occurred. Read-only.

Syntax

Error.Column

Remarks

Use the **Column** property with the **Line** property to determine the approximate location of the scripting code causing an error. If no column information is available, the **Column** property is set to zero.

Note Not all **Error** object properties are valid at all times. When a property is not appropriate for a specific error, its value will be 0 or a zero-length string ("").

Count Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproCountC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproCountX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproCountA"}

Returns the number of items in a collection. Read-only

Syntax

object.**Count**

The *object* is any collection in the "Applies To" list or a reference to it.

Description Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproDescriptionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproDescriptionX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproDescriptionA"}

Returns the short descriptive string associated with an error. Read-only.

Syntax

Error.Description

Remarks

Use the **Description** property to alert a user to an error that you either can't or don't want to handle. The **Description** property provides the built-in system error messages.

Note Not all **Error** object properties are valid at all times. When a property is not appropriate for a specific error, its value will be 0 or a zero-length string ("").

Error Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproErrorX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproErrorA"}

Returns an **Error** object containing detailed information about the last error that occurred. Read-only.

Syntax

ScriptControl.Error

Remarks

The returned **Error** object is used for both compilation and run-time errors.

Note Not all **Error** object properties are valid at all times. When a property is not appropriate for a specific error, its value will be 0 or a zero-length string ("").

HasReturnValue Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproHasReturnValueC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproHasReturnValueX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproHasReturnValueA"}

Returns **True** if a procedure has a return value; **False** if it does not. Read-only.

Syntax

object.**HasReturnValue**

The *object* is the name of a specific **Procedure** or reference to it.

HelpContext Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproHelpContextC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproHelpContextX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproHelpContextA"}

Returns the context ID number for a topic in a Microsoft Windows Help file. Read-only.

Syntax

Error.HelpContext

Remarks

The **HelpContext** property is used to automatically display the Help topic specified by the **HelpFile** property.

Note You should write routines in your application to handle typical errors. When programming with an object, you can use the object's Help file to improve the quality of your error handling, or to display a meaningful message to your user if the error isn't recoverable.

Note Not all **Error** object properties are valid at all times. When a property is not appropriate for a specific error, its value will be 0 or a zero-length string ("").

HelpFile Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproHelpFileC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproHelpFileX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproHelpFileA"}

Returns a string expression that is the fully qualified path to a Microsoft Windows Help file. Read-only.

Syntax

Error.HelpFile

Remarks

If a Help file is specified in the **HelpFile** property, it is automatically called when the user clicks the **Help** button (or presses the F1 key) in the error message dialog box. If the **HelpContext** property contains a valid context ID for the specified file, that topic is automatically displayed. If no **HelpFile** is specified, the Visual Basic Help file is displayed.

Note You should write routines in your application to handle typical errors. When programming with an object, you can use the object's Help file to improve the quality of your error handling, or to display a meaningful message to your user if the error isn't recoverable.

Note Not all **Error** object properties are valid at all times. When a property is not appropriate for a specific error, its value will be 0 or a zero-length string ("").

Language Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproLanguageC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproLanguageX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproLanguageA"}

Sets or returns the name of the scripting language being used. Read/write.

Syntax

ScriptControl.Language [= *language*]

When setting the **Language** property, *language* can be VBScript, JScript, or any other suitable scripting language.

Line Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproLineC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproLineX":1}
{ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproLineA"}

Returns a line location indicating the place in scripting code where an error occurred. Read-only.

Syntax

Error.Line

Remarks

Use the **Line** property with the **Column** property to determine the approximate location of the scripting code causing an error. If no line information is available, the **Line** property is set to zero.

Note Not all **Error** object properties are valid at all times. When a property is not appropriate for a specific error, its value will be 0 or a zero-length string ("").

Modules Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproModulesC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproModulesX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproModulesA"}

Returns the collection of modules for the **ScriptControl** object. Read-only.

Syntax

ScriptControl.Modules

Remarks

There is always at least one module. If a module is not explicitly declared, an implicit **Global** module exists that is used for all added scripting code. The **Global** module can be accessed using the **GlobalModule** constant as an index to the **Modules** collection.

Name Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproNameC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproNameX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproNameA"}

Returns the name of a module, procedure, or object. Read-only.

Syntax

object.**Name**

The *object* is any object in the "Applies To" list or a reference to it.

Remarks

The name of *object* is established at the time *object* is added to the **ScriptControl** using **Add** or **AddObject** methods.

NumArgs Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproNumArgsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproNumArgsX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproNumArgsA"}

Returns the number of arguments required by a procedure. Read-only.

Syntax

object.**NumArgs**

The *object* is a **Procedure** object or a reference to it.

Number Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproNumberC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproNumberX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproNumberA"}

Returns a numeric value specifying a run-time error. **Number** is the **Error** object's default property. Read-only.

Syntax

Error.Number

Remarks

Error numbers only apply to run-time errors.

Procedures Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproProceduresC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproProceduresX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproProceduresA"}

Returns the collection of the procedures defined in a specified module. Read-only.

Syntax

object.**Procedures**

The *object* is a **Module** object or a reference to it.

SitehWnd Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproSitehWndC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproSitehWndX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproSitehWndA"}

Sets or returns the hWnd of the window used by executing scripting code to display dialog boxes and other user-interface elements. Read/Write.

Syntax

ScriptControl.SitehWnd [= *value*]

The *value* argument can be 0 or the number of a valid hWnd.

Remarks

If the **ScriptControl** is being used as a control (rather than an Automation object), the default value of the **SitehWnd** property is the hWnd of the control's container. If **ScriptControl** is created as an Automation object, the default **SitehWnd** is 0, which indicates the desktop window. In either case, the user can change the value to a valid hWnd at any time.

Source Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproSourceC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproSourceX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproSourceA"}

Returns a string expression specifying the type of error that occurred. Read-only.

Syntax

Error.Source

Remarks

The string expression returned by the **Source** property. specifying the kind of error generated can be any of the following:

- Microsoft JScript™ compilation error.
- Microsoft JScript run-time error.
- Microsoft VBScript compilation error.
- Microsoft VBScript run-time error.

State Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproStateC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproStateX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproStateA"}

Sets or returns the mode of the **ScriptControl** object. Read/write.

Syntax

ScriptControl.State [= *state*]

The *state* argument can be either of the following constants:

Constant	Value	Description
Initialized	0	The scripting engine will execute the code but will not sink any events generated by objects added using the **AddObject** method.
Connected	1	The scripting engine will sink events generated by objects added using the **AddObject** method.

Text Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproTextC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproTextX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproTextA"}

Returns a string containing a snippet of the scripting code surrounding the location where an error occurred in code. Read-only.

Syntax

Error.Text

Remarks

The string of scripting code returned by the **Text** property serves as the context in which an error occurred.

Timeout Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproTimeOutC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproTimeOutX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproTimeOutA"}

Sets or returns the time, in milliseconds, after which a user is presented with the option to discontinue scripting code execution or allow it to continue. Read/write.

Syntax

ScriptControl.Timeout [= *value*]

If *value* is specified using the **NoTimeout** constant (-1), no timeout is used. When value is 0, the Timeout event occurs as soon as it is determined that a running script is hung for any reason.

Remarks

When a timeout occurs, **ScriptControl** checks the **AllowUI** property on all running **ScriptControl** objects to see whether it is permitted to display user-interface elements.

If the **AllowUI** property is **True** for any of the running controls, when a timeout occurs, the user is alerted with a dialog box. Selecting **End** causes causes the **ScriptControl** to stop the execution of all scripting engines until all engines have been removed from the call stack. Selecting **Continue** causes scripting code execution to continue for another timeout period based on the minimum value of the **Timeout** property for all currently running scripting engines. At the end of each timeout period, the dialog box is displayed, and user action is again required.

If the **AllowUI** property is **False** when a timeout occurs, the script execution is immediately ended, and the user receives no notification of what happened.

UseSafeSubset Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scproUseSafeSubsetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scproUseSafeSubsetX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scproUseSafeSubsetA"}

Sets or returns a Boolean value indicating whether the host application is concerned about safety. **UseSafeSubset** is **True** if the host application cares about safety; **False** if it does not. Read/write.

Syntax

ScriptControl.UseSafeSubset [=*value*]

The **UseSafeSubset** property has these parts:

If *value* is **True**, access to some objects and procedures is not allowed.

Remarks

The scripting engine in use determines if, and when, the **UseSafeSubset** property is set. If no language is specified (**Language** property), the **UseSafeSubset** can be set at any time.

The objects and procedures that can't be used when **UseSafeSubset** is **True** are identical to those restricted by the browser's highest security setting.

Add Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scmthAddC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scmthAddX":1}
{ewc HLP95EN.DLL,DYNALINK,"Applies To":"scmthAddA"}

Adds a new module to the **Modules** collection.

Syntax

ScriptControl.Modules.Add(n*ame*[, *object*])

The **Add** method has these parts:

Part	Description
name	Required. String name of the module being added.
object	Optional. The name of an object associated with the module.

Remarks

Optionally, an added module can have an object associated with it. If this kind of object is specified, event-handling code can be written behind the object and its subordinate objects.

AddCode Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scmthAddCodeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scmthAddCodeX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scmthAddCodeA"}

Adds specified code to a module.

Syntax

object.**AddCode** *code*

The **AddCode** method has these parts:

Part	Description
object	Required. The object to which code is being added or a reference to it. Can be any object in the "Applies To" list.
code	Required. String containing code being added to the object.

Remarks

AddCode may be called multiple times.

AddObject Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scmthAddObjectC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scmthAddObjectX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scmthAddObjectA"}

Makes run-time functionality available to a scripting engine.

Syntax

ScriptControl.AddObject(*name*, *object*[, *addMembers*])

The **AddObject** method has these parts:

Part	Description
name	Required. Name by which the added object is to be known in **ScriptControl** code.
object	Required. Name of the object exposed at run time.
addMembers	Optional. Boolean value. **True** if members of *object* are globally accessible; **False** if they are not.

Remarks

Use the **AddObject** method to make run-time functionality available to a scripting engine. The **AddObject** method enables a **ScriptControl** user to provide a set of *name/object* pairs to the scripting code. The scripting engines may expose the *name* in any way. In both VBScript and JScript, each *name* appears as a globally accessible name.

Clear Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scmthClearC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scmthClearX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scmthClearA"}

Clears all **Error** object properties.

Syntax

Error.Clear

Remarks

The properties of the **Error** object are routinely cleared when **Reset**, **AddCode**, **Eval**, or **ExecuteStatement** methods are executed. The **Clear** method is used in all other circumstances to clear **Error** object properties.

Eval Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scmthEvalC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scmthEvalX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scmthEvalA"}

Evaluates an expression and returns the result.

Syntax

object.**Eval**(*expression*)

The **Eval** method has these parts:

Part	Description
object	Required. The object in which the expression is being evaluated or a reference to it. Can be any object in the "Applies To" list.
expression	Required. String containing the expression being evaluated.

Remarks

The context of the **Eval** method is determined by the *object* argument. If *object* is a module, the context is restricted to the named module. If *object* is the **ScriptControl**, the context is global.

ExecuteStatement Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scmthExecuteStatementC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scmthExecuteStatementX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scmthExecuteStatementA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"scmthExecuteStatementS"}

Executes a specified statement.

Syntax

object.**ExecuteStatement** *statement*

The **ExecuteStatement** method has these parts:

Part	Description
object	Required. The object providing the context in which the statement is being evaluated or a reference to it. Can be any object in the "Applies To" list.
statement	Required. String containing the statement being executed.

Remarks

The context of the **ExecuteStatement** method is determined by the *object* argument. If *object* is a module, the context is restricted to the named module. If *object* is the **ScriptControl**, the context is global.

Item Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scmthItemC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scmthItemX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scmthItemA"}

Returns a specific member of a collection either by position or by key.

Syntax

object.**Item(***index***)**

The **Item** method has these parts:

Part	Description
object	Required. Can be any collection in the "Applies To" list or a reference to it.
index	Required. An expression that specifies the position of a member of the collection. If a numeric expression, *index* must be a number from 1 to the value of the collection's **Count** property. If a string expression, *index* must correspond to the name of the member as specified when the member was added to the collection. Use the **GlobalModule** constant to access the special **Global** module that is always present in the **Modules** collection.

Remarks

In general, the order of members of a collection can't be relied on to be static. However, between additions and deletions, members do not move around.

Reset Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scmthResetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scmthResetX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scmthResetA"}

Discards all scripting code and objects that have been added to the **ScriptControl**.

Syntax

ScriptControl.Reset

Remarks

The **State** property of the **ScriptControl** is reset to **Initialized** (0).

Run Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scmthRunC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scmthRunX":1}
{ewc HLP95EN.DLL,DYNALINK,"Applies To":"scmthRunA"}

Runs a specified procedure.

Syntax

object.**Run**(*procedureName*, *parameters()*)

The **Run** method has these parts:

Part	Description
object	Required. The object in which the procedure is being run or a reference to it. Can be any object in the "Applies To" list.
procedureName	Required. String name of the procedure to run.
parameters()	Required. An array containing any parameters for the procedure being run.

Remarks

There are two ways to run a procedure:

- Use the **Run** method. To do this, specify the procedure name as a string. This is useful when the names of the procedures are not known in advance.
- Use the **CodeObject** and call it directly. This is useful when you know the names of all procedures ahead of time.

Error Event

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scevtErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scevtErrorX":1}
{ewc HLP95EN.DLL,DYNALINK,"Applies To":"scevtErrorA"}

Occurs in response to a run-time error.

Syntax

Sub Error(*error*)

The *error* argument is the number of the run-time error that occurred.

Remarks

An Error event can occur during event sinking or during a direct call to the scripting engine when calling into an object returned by the **CodeObject** property.

Timeout Event

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scevtTimeOutC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"scevtTimeoutX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"scevtTimeOutA"}

Occurs when the time specified in the **Timeout** property has elapsed, and a user has selected **End** from the resulting dialog box.

Syntax

Sub Timeout

Remarks

If several **ScriptControl** objects are present, the Timeout event will only occur for the initial **ScriptControl** object.

ScriptControl Constants

{ewc HLP95EN.DLL,DYNALINK,"See Also":"sccstScriptControlC"}

The **ScriptControl** Object uses two different categories of constants: constants that apply to the **ScriptControl** library and constants that define **ScriptControl** states.

ScriptControl Constants

The constants used by the **ScriptControl** object library are defined as follows:

Constant	Value	Description
GlobalModule	Global	Scripting engines support at least one module called the **Global** module, which is created when the **ScriptControl Language** property is set to a valid language.**GlobalModule** is used as an index into the **ScriptControl Modules** collection to retrieve this special module.
NoTimeout	-1	Used to set the **ScriptControl Timeout** property so that scripting engine execution never times out.

ScriptControl States

The constants used to indicate **ScriptControl** states are defined as follows:

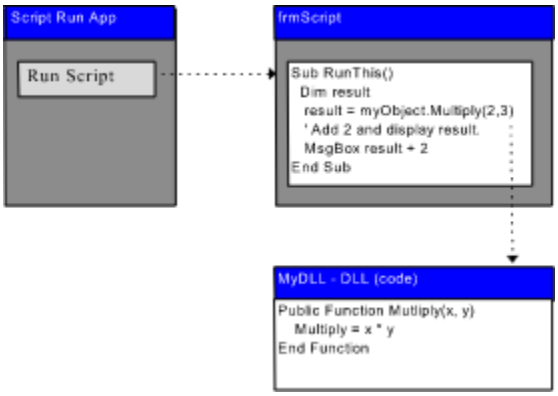
Constant	Value	Description
Initialized	0	The scripting engine is initialized. The scripting engine will execute code but will not sink events generated by any objects added using the **AddObject** method.
Connected	1	The scripting engine will sink events generated by objects added using the **AddObject** method.

Adding Run-time Functionality to the Script Control

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scconAddingRuntimeFunctionalityToScriptControlC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"scconAddingRuntimeFunctionalityToScriptControlS"}

A major feature of the Script Control is the ability to add the run-time functionality of an ActiveX component to the control. Run-time functionality includes the public methods and properties of an object. The object can be any ActiveX component, such as an ActiveX DLL that has such public methods and properties. In other words, the Script Control can behave as a macro engine to invoke the methods of a component or set the component's properties.

The following figure shows a simple implementation of this concept.



An application containing the Script Control runs a script contained on the form named `frmScript`. The script, written in VBScript, invokes the `Multiply` function which is defined in the component named `MyObject.dll`. The script assigns the result of the function to a variable, adds 2, and displays the final result.

Using the AddObject Method

The syntax of the **AddObject** method requires a string and the object to be added. The string is the name of object when the object is invoked in a script. For example, the following code fragment adds a class module named `MyClass` to the Script Control. The name of the object is set to `MyScriptObject`:

```
Option Explicit
' Declare the object variable for the MyClass object.
Private objMyClass As New MyClass

Private Sub Form_Load()
' The Name of the Script Control is scDemo.
   scDemo.AddObject "MyScriptObject", objMyClass
End Sub
```

When you subsequently write a script that invokes the object's methods or properties, the script must use the name `MyScriptObject`. For example, if the class module has a public function named `TaxIt`, and a public property named `Rate`, the script might contain the following code:

```
' This is VBScript code.
Sub RunObject()
   Dim NewRate
   NewRate = InputBox("New rate:",,.09)
   ' Set the object's property.
   MyScriptObject.Rate = NewRate
   Dim price
```

```
   price = InputBox("Price:",,100)
   ' Perform an operation using the TaxIt function.
   MsgBox MyScriptObject.TaxIt(price)
End Sub
```

Use the **Run** method to run the script:

```
scDemo.Run "RunObject"
```

Creating a Simple Script Project

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scconCreatingSimpleScriptProjectC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"scconCreatingSimpleScriptProjectS"}

To demonstrate the addition of run-time functionality to the Script Control, use the following steps to construct a sample Visual Basic project named `SimpleScript`.

Create the SimpleScript Project

1. On the **File** Project, click **New Project**.
2. In the **New Project** dialog box, double-click the **Standard Exe** icon.
3. Press F4 to open the **Properties** window. Double-click the **Name** property and change the form name to frmSimpleScript.
4. Double-click the **Caption** property and set it to "Simple Script Demo."
5. On the **Project** menu, click **Add Class Module**.
6. In the **Add Class Module** dialog box, double-click the **Class Module** icon. You will add the run-time functionality of this object to the Control.
7. In the **Properties** window, double-click **Name** and change the name to "MyClass."
8. Add the following code to the class module:

```
Option Explicit
Private ObjRate As Double

Public Property Get Rate() As Double
    Rate = ObjRate
End Property

Public Property Let Rate(newRate As Double)
    ObjRate = newRate
End Property

Public Function TaxIt(price) As Double
    TaxIt = price + (price * ObjRate)
End Function
```

9. Add the following controls to the form, and set their properties to the values listed in the following tables.

CommandButton1

Property	Value
Name	cmdAddCode
Caption	AddCode
Height	255
Left	0
Top	240
Width	1215

TextBox1

Property	Value
Name	txtScript
MultiLine	True
ScrollBars	2 - Vertical
Height	1575
Left	1320

Top	240
Width	4335

ListBox1

Property	Value
Name	lstProcedures
Height	645
Left	1320
Top	2160
Width	4335

10 Right-click the Toolbox and click **Components** to display the **Add Component** dialog box. Double-click **Microsoft Script Control 5.0**, and then click **OK**.

11 On the Toolbox, double-click the **ScriptControl** icon to add it to the form.

12 Rename the Script Control `scDemo`.

13 Paste the following code into the Declaration section of the form.

```
Option Explicit
Private MyObject As New MyClass

Private Sub Form_Load()
   ' Add MyObject to the Script Control. This adds
   ' the run-time functionality of the object to the
   ' control.
   scDemo.AddObject "objScript", MyObject
   ' Add script to Textbox control.
   txtScript.Text = _
   "Sub SetRate()" & vbCrLf & _
   "  Dim Rate" & vbCrLf & _
   "  Rate = InputBox(""TaxRate:"",,.086)" & _
   vbCrLf & _
   "  objScript.Rate = Rate" & vbCrLf & _
   "End Sub" & vbCrLf & vbCrLf & _
   "Sub TotalPrice()" & vbCrLf & _
   "  Dim price, ttl" & vbCrLf & _
   "  price = InputBox(""Price:"",,100)" & _
   vbCrLf & _
   "  ttl =objScript.TaxIt(price)" & vbCrLf & _
   "  MsgBox ttl" & vbCrLf & _
   "End Sub"
End Sub

Private Sub cmdAddCode_Click()
   ' Add the code in the TextBox to the control.
   scDemo.AddCode txtScript.Text

   ' Clear the ListBox, and add the name of each
   ' procedure in the Procedures collection.
   lstProcedures.Clear
   Dim p As Procedure
   For Each p In scDemo.Procedures
      lstProcedures.AddItem p.Name
   Next
```

```
End Sub

Private Sub lstProcedures_Click()
   ' Run the procedure in the ListBox.
   scDemo.Run lstProcedures.Text
End Sub
```

Run the SimpleScript Project

1. Press F5 to run the project.
2. Click **AddCode** to add the code to the control. The ListBox control is filled with the names of the procedures that were just added.
3. In the ListBox control, click **SetRate**. A dialog box displays the default value 8.6. Change the rate or just click **OK**.
4. In the ListBox control, click **TotalPrice**. A dialog box displays the default value 100. Change the value and click **OK**. The total of the value plus the tax rate will then be displayed.

Editing Code in the Textbox Control

You can also change the code before adding it to the control again.

1. In the TextBox control, change the **SetRate** default value for the MsgBox to 4.2.
2. Click **AddCode**.
3. In the ListBox control, click **SetRate**. The default has now changed.

Adding a Procedure in the TextBox Control

Experiment by inserting your own code into the Textbox control.

1. In the Textbox control, paste the following code:

```
Sub VariableTax()
   Dim Years, results, ttlInterest
   Dim i, principal, payment, interest, mortgage
   principal = InputBox("Borrow:",,100000)
   Years = InputBox("How long:",,15)
   results = "Payment" & vbtab & "Interest" & vbcrlf
   For i = 1 to Years * 12
    interest =  (principal * objScript.Rate) / 12
    principal = principal - interest
    ttlInterest = ttlInterest + interest
    results = results & i & vbtab & formatNumber(interest,2) & vbcrlf
    Next
    MsgBox results
    MsgBox "Total Interest Paid:" & "$" & formatNumber(ttlInterest,2)
  End Sub
```

2. Click **AddCode**.
3. Click **SetRate**, change the rate, and then click **OK**.
4. Click **VariableTax** to run the new procedure.

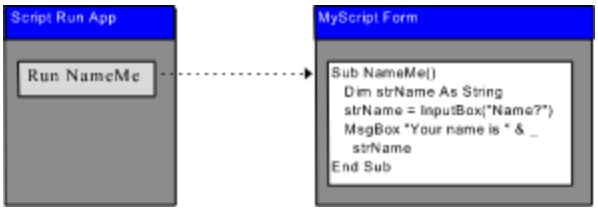
Using the Microsoft Script Control

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scconUsingMicrosoftScriptControlC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"scconUsingMicrosoftScriptControlS"}

The Microsoft Script Control allows the user to write and run scripts for a any scripting engine, such as VBScript or JScript, both of which are included with the Script Control. You can add the run-time functionality of any Automation object that exposes methods and properties to the Script Control. By combining the run-time functionality of an application with a scripting engine, you can create a scripting application that runs macros to control the application.

Overview

The Microsoft Script Control allows you to create an application that runs a scripting language such as VBScript or JScript. For example, the following diagram illustrates a simple implementation. When the user clicks the `Run NameMe` button, the `NameMe` procedure on the `MyScript` form is executed by the VBScript scripting engine.



The `Run NameMe` button uses the Script Control's **Run** method to execute the script.

```
' The name of the Script Control is ScriptControl1.
Private Sub RunNameMe_Click()
   ScriptControl1.Run "NameMe"
End Sub
```

You can use the following steps to create the rest of the code required for this script:

1. Select a scripting language.
2. Add code to a procedure.
3. Run the procedure.

Select a Scripting Language

The first step is to configure the Script Control for the correct scripting language. By default, the Script Control uses VBScript. Use the **Properties** window to set the language property JScript, if that is appropriate. You can also use the **Language** property in code, as shown below:

```
ScriptControl1.Language = "JScript"
```

Other scripting languages can be used by the Script Control.

Add Code to a Procedure

Before you can run the `NameMe` procedure, use the **AddCode** method to add the complete procedure to the Script Control. If you try to add an incomplete procedure (one with no **End Sub** or **End Function**), an error will occur. The following example adds procedure code to the Script Control:

```
' When the ScriptRun app loads, the following code
' adds the NameMe procedure to the Control.
Private Sub Form_Load()
   Dim strCode As String
   strCode = _
   "Sub NameMe()" & vbCrLf & _
   " Dim strName As String" & vbCrLf & _
```

```
   " strName = InputBox(""Name?"")" & vbCrLf & _
   " MsgBox ""Your name is "" & strName" & vbcrLf & _
   "End Sub"
   ScriptControl1.AddCode strCode
End Sub
```

Alternatively, you can add procedure code using a TextBox control:

```
Private Sub Form_Load()
   ' The code is contained in the Textbox named
   ' txtScript on the form named frmScript.
   ScriptControl1.AddCode frmScript.txtScript.Text
End Sub
```

You can add arguments to a procedure or function.

```
Private Sub EvalFunc()
   ' Create the function.
   Dim strFunction As String
   strFunction = _
   "Function ReturnThis(x, y)" & vbCrLf & _
   " ReturnThis = x * y" & vbCrLf & _
   "End Function"
   ' Add the code, then run the function.
   ScriptControl1.AddCode strFunction
   MsgBox ScriptControl1.Run("ReturnThis", 3, 25)
End Sub
```

Once code has been added using the **AddCode** method, you can then use the **Run** method to run the procedure.

Run the Procedure

The **Run** method runs any complete procedure that has been added to the Script Control. The following code fragment runs three defined procedures:

```
ScriptControl1.Run "FindName"
ScriptControl1.Run "AddName"
ScriptControl1.Run "Quit"
```

Executing Scripting Statements

To execute scripting code, the Script Control features the **ExecuteStatement** method. For example, the following code executes a **MsgBox** statement:

```
Private Sub Command1_Click()
   ' Create a message box with the word Hello in it.
   ScriptControl1.ExecuteStatement "MsgBox ""Hello"""
End Sub
```

Evaluating Scripting Statements

You can also evaluate lines of scripting code using the **Eval** method. The **Eval** method simply tests a line of scripting code, as shown in the following example:

```
Private Sub TryThis()
    ScriptControl1.ExecuteStatement "x = 100"
    MsgBox ScriptControl1.Eval("x = 100")  ' True
    MsgBox ScriptControl1.Eval("x = 100/2")  ' False
End Sub
```

In the preceding code, the **ExecuteStatement** method executes the statement and assigns the value 100 to the variable x. The next two lines use the **Eval** method to test the statements x = 100 and x =

100/2. The second line returns **True**, and the third returns **False**.

Creating an Instance of the Script Control

The Microsoft Script Control can be created as either a control or a standalone Automation object. This feature allows the Script Control to be used by any host, using any scripting language. The following example can be placed in any form. Note that the variable `sc` is not declared as type **ScriptControl** because the Control is not—and doesn't need to be—referenced in the project. As long as the Script Control is present and registered, the following code will work:

```
Private Sub Command1_Click()
   Dim sc ' This can't be early-bound!
   Dim strProgram As String
   strProgram = "Sub Main" & vbCrLf & _
   "MsgBox ""Hello World""" & vbCrLf & _
   "End Sub"
   Set sc = CreateObject("ScriptControl")
   sc.language = "VBScript"
   sc.addcode strProgram
   sc.run "Main"
End Sub
```

Using the AllowUI Property

The **AllowUI** property determines if the scripting engine is permitted to display user interface elements. This can apply to the Script Control itself, such as when it displays timeout messages. It can also apply to scripting engines that use ActiveX scripting interfaces. For example, the following code will generate an error:

```
ScriptControl1.AllowUI = False
Dim strX As String
strX = "Sub Hello" & vbCrLf & _
"MsgBox ""Hello, World""" & vbCrLf & _
"End Sub"
ScriptControl1.AddCode strX
ScriptControl1.Run "Hello" ' No UI allowed!
```

Using the Error Property

With the Script Control, errors may come from two sources: the Script Control itself, or the script the Control is attempting to run. In the latter case, to debug scripting code, the Script Control features the **Error** property, which returns a reference to the **Error** object. Using the **Error** object, the Script Control can return the number and description of the error, and also the the line number where the error occurs in the script.

Run the following code to see an example of the Control finding an error:

```
Private Sub MyError()
   ' The scripting code below divides by zero raising
   ' an error.
   Dim strCode As String
   strCode = _
   "Sub DivideByZero()" & vbCrLf & _
   "Dim prime" & vbCrLf & _
   "prime = 3" & vbCrLf & _
   "MsgBox prime/0" & vbCrLf & _
   "End Sub"
   On Error GoTo scError
   With ScriptControl1
      .AddCode strCode
```

```
      .Run "DivideByZero"
   End With
   Exit Sub
scError:
   ' Use the Error object to inform the user of the
   ' error, and what line it occured in.
   Debug.Print ScriptControl1.Error.Number & _
   ":" & ScriptControl1.Error.Description & _
   " in line " & ScriptControl1.Error.Line
   Exit Sub
End Sub
```

Using the Modules Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"scconUsingModulesCollectionC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"scconUsingModulesCollectionS"}

One way to manage your scripts is to use the **Module** object and the **Modules** collection.

Using Modules and Procedures

In Visual Basic, code is written in modules. Each module contains a set of functions and procedures that are related in a logical manner. Each **Module** object in the **Modules** collection has a one-to-one correlation to each script that you want to manage. Each **Module** object contains a collection of **Procedure** objects, each of which can be run using the **Run** method.

Adding Procedures to the Global Module

By default, the Script Control contains one Module, the **Global** module. All procedures contained in the **Global** module can be invoked by other modules. For example, the following code will return the name of the default module (**Global**):

```
' The Script Control is named scDemo.
MsgBox scDemo.Modules(1).Name ' returns "Global"
```

If you are working with only one Module, you can simply add code to the **Global** module by invoking the **AddCode** method without specifying a module, as shown below:

```
' Add the code contained in a textbox control named
' txtMyScript.
scDemo.AddCode txtMyScript.Text
```

Similarly, you can run any code that you added to the default module by invoking the **Run** method on the Script Control:

```
' Run a procedure named RunThis.
scDemo.Run "RunThis"
```

Adding Modules and Procedures

As with any collection object, you can use the **Add** method to add to the collection, giving each Module a name as you add it. Subsequently, you can use that name to specify the Module object, and call its procedures.

The following example first adds a module to the collection naming it `Numbers`. The code then adds the procedures found in a Textbox control to the **Module** object in the collection. Finally, a procedure from the module is run.

```
scDemo.Modules.Add "Numbers"
scDemo.Modules("Numbers").Procedures.Add _
   txtMyScript.Text
' Run a procedure named Factors.
scDemo.Run scDemo.Modules("Numbers"). _
Procedures("Factors")
```

Clearing the Modules Collection

To clear all **Module** objects from the **Modules** collection, use the **Reset** method. The collection's **Count** property will be reset to 1, with only the Global module object remaining. All procedures (including those stored in the Global module) will also be cleared.

Using the CodeObject Property

After you have populated a **Module** object with procedures, you can use the **CodeObject** property to directly invoke a procedure instead of using the **Run** method. The following code shows an example

of this:

```
Dim modX As Module
Set modX = scDemo.Modules.Add("myMod")
Dim strX As String
strX = "Sub Hello" & vbCrLf & _
"MsgBox ""Hello, World""" & vbCrLf & _
"End Sub"

modX.AddCode strX
Dim objX As Object
' Set the variable to the CodeObject, then
' invoke the procedure.
Set objX = modX.CodeObject
objX.Hello
```