Windows Script Interfaces

By mikejo5000

Archived: 2026-04-05 14:04:22 UTC

Microsoft Windows Script Interfaces provide a way for an application to add scripting and OLE automation. Scripting hosts that rely on Windows Script can use scripting engines from multiple sources and vendors to manage scripting between components. The implementation of the script itself—language, syntax, persistent format, execution model, and so on—is left to the script vendor.

Windows Script documentation is divided into the following sections:

Windows Script Hosts

Windows Script Engines

Active Script Debugging Overview

Active Script Profiling Overview

Windows Script Interfaces Reference

Windows Script Background

Windows Script interfaces fall into two categories: Windows Script hosts and Windows Script engines. A host creates a scripting engine and calls on the engine to run the scripts. Examples of Windows Script hosts include:

- Microsoft Internet Explorer
- Internet authoring tools
- Shell

Windows Script engines can be developed for any language or run-time environment, including:

- Microsoft Visual Basic Scripting Edition (VBScript)
- Perl
- Lisp

To make implementation of the host as flexible as possible, an OLE Automation wrapper for Windows Script is provided. However, a host that uses this wrapper object to instantiate the scripting engine does not have the degree

of control over the run-time name space, the persistence model, and so on, that it would if it used Windows Script directly.

The Windows Script design isolates the interface elements required only in an authoring environment so that nonauthoring hosts (such as browsers and viewers) and script engines (for example, VBScript) can be kept lightweight.

Windows Script Basic Architecture

The following illustration shows the interaction between a Windows Script host and an Windows Script engine.

The steps involved in the interaction between the host and engine are given in the following list.

1. Create a project. The host loads a project or document. (This step is not particular to Windows Script, but is included here for completeness.)

2. Create the Windows Script engine. The host calls `CoCreateInstance` to create a new Windows Script engine, specifying the class identifier (CLSID) of the specific scripting engine to use. For example, the HTML browser of Internet Explorer receives the scripting engine's class identifier through the CLSID= attribute of the HTML <OBJECT> tag.

3. Load the script. If the script contents have been persisted, the host calls the script engine's `IPersist*::Load` method to feed it the script storage, stream, or property bag. Otherwise, the host uses either the `IPersist*::InitNew` or IActiveScriptParse::InitNew method to create a null script. A host that maintains a script as text can use IActiveScriptParse::ParseScriptText to feed the scripting engine the text of the script, after calling `IActiveScriptParse::InitNew`.

4. Add named items. For each top-level named item (such as pages and forms) imported into the scripting engine's name space, the host calls the IActiveScript::AddNamedItem method to create an entry in the engine's name space. This step is not necessary if top-level named items are already part of the persistent state of the script loaded in step 3. A host does not use `IActiveScript::AddNamedItem` to add sublevel named items (such as controls on an HTML page); rather, the engine indirectly obtains sublevel items from top-level items by using the host's `ITypeInfo` and `IDispatch` interfaces.

5. Run the script. The host causes the engine to start running the script by setting the SCRIPTSTATE_CONNECTED flag in the IActiveScript::SetScriptState method. This call would likely perform any scripting engine construction work, including static binding, hooking up to events (see below), and executing code, in a way similar to a scripted `main()` function.

6. Get item information. Each time the script engine needs to associate a symbol with a top-level item, it calls the IActiveScriptSite::GetItemInfo method, which returns information about the given item.

7. Hook up events. Before starting the actual script, the scripting engine connects to the events of all the relevant objects through the `IConnectionPoint` interface.

8. Invoke properties and methods. As the script runs, the scripting engine realizes references to methods and properties on named objects through `IDispatch::Invoke` or other standard OLE binding mechanisms.

Windows Script Terms

This list contains definitions of the scripting-related terms used in this document.

Term	Definition
Code object	An instance created by the scripting engine that is associated with a named item, such as the module behind a form in Visual Basic, or a C++ class associated with a named item. Preferably, this is an OLE Component Object Model (COM) object that supports OLE Automation so the host or other non-script entity can manipulate the code object.
Named item	An OLE COM object (preferably one that supports OLE Automation) that the host deems interesting to the script. Examples include the HTML Page and Browser in a Web browser, and the Document and Dialogs in Microsoft Word.
Script	The data that makes up the program that the scripting engine runs. A script can be any contiguous executable data, including pieces of text, blocks of `pcode`, or even machine-specific executable byte codes. A host loads a script into the scripting engine through one of the `IPersist*` interfaces or through the IActiveScriptParse interface.
Scripting engine	The OLE object that processes scripts. A scripting engine implements the IActiveScript and, optionally, IActiveScriptParse interfaces.
Scripting host	The application or program that owns the Windows Script engine. The host implements the IActiveScriptSite and, optionally, IActiveScriptSiteWindow interfaces.
Scriptlet	A portion of a script that gets attached to an object through the IActiveScriptParse interface. The aggregate collection of scriptlets is the script.
Script language	The language in which a script is written (VBScript, for example) and the semantics of that language.
