“APPLYING THE OBJECT REXX WINDOWS SCRIPT ENGINE WITH THE WINDOWS SCRIPT HOST”

Rony G. Flatscher

Department of Management Information Systems/Business Informatics at the

Vienna University of Economics and Business Administration (Austria)

ABSTRACT

The Windows Script Host (WSH) has been part of the Microsoft Windows operating systems since Windows 98 (16 Bit version) and Windows 2000 (32 Bit version). For all other Windows operating systems (and updates as well) the WSH is installed with Microsoft Internet Explorer, which depends on that technology. Extending the abilities of OLE/ActiveX automation WSH allows for automating/remote controlling Windows itself as well as applications serving as Windows script hosts like Microsoft's Internet Explorer.

By default WSH comes with the programming languages JScript (Microsoft's JavaScript/ECMAScript implementation) and Visual Basic Script edition (VBScript). In addition, any programming language which adheres to the WSH specifications may be used to interact with automatable applications, hence the very same functionality can be achieved e.g. by IBM's Object Rexx for Windows.

This paper introduces the WSH architecture from a conceptual point of view, discusses its applications and implications for companies using Windows, and in this context stresses the most important security issues which adhere to WSH.

1 INTRODUCTION

The term “script” denotes a mechanism by which the commands of human beings are collected and structured by the means of an electronic file, which then serves as the input to the operating and/or application system. The first scripts were created many decades ago for mainframe class computers, where recurrent commands entered by the operators were instead stored in a file as a “batch of commands”[1)] to be carried out in exactly the given sequence. Usually an operating system program carried out the content of such a batch file.

In the meantime this principle has been applied to application systems as well, such as Corel Draw, Lotus Smartsuite, Microsoft Office and the like, but are usually called “macros” instead of “scripts”. Even endusers in business departments are able to create such scripts[2)] and have them executed with the built in macro facilities.

In addition to merely storing a set of recurring commands for sequential execution, simple programming languages were created, which could react upon certain conditions as a result of carrying out single commands out of a file, such that programmatically one could decide which commands should be carried out under which given circumstances. Such programming languages are called “batch”, “command”, “script” or “macro” languages, industrial strength examples for such scripting languages include (but are not restricted to): JavaScript, Perl, REXX, TcL, VBScript.

---

1) That is the reason why under DOS such files have been called “batch files” and have received the file type “.BAT”. On the Windows platform these type of files have been renamed to “command files” indicated by the file type “.CMD”.

2) “Scripts” or “macros” in this context are usually created by having the macro facility remember all key-strokes the user enters. After ending the recording of key-strokes, representing commands and input data, the end-user is able to “replay” all the stored key-strokes at their will at any time.

2 SCRIPTING UNDER WINDOWS

In the Windows environment Microsoft introduced at the end of the 80ies the technology “Object Linking and Embedding” (OLE), which later was extended and renamed to “ActiveX”. At the core of this technology the definition of “components” became possible which possess a minimal interface which can be used by other programs to explore and inspect additional, application supplied (i.e. non standardized) interfaces to it. The Microsoft “component object model” (COM) defines a minimal set of these interfaces allowing for this exploration and invoking found programming interfaces[3]. With this building stone it has become possible to interact with Windows-based components from different processes within the same physical Windows machine and has been since extended to be applicable over networks, such that disparate Windows machines could use each other’s functionality via Microsoft’s “distributed COM” (DCOM) technology.

In the course of the last decade Microsoft defined interfaces for COM-based applications, with the explicit intent for those COM programs to become scriptable, i.e. to allow for sending commands to them in a standardized way which makes this technology[4] feasible to be even used from (even fairly simple) scripting languages.

Figure 1 depicts a little (self-explaining) Object Rexx script, which remote controls the Microsoft Internet Explorer (MSIE). This is possible, because a) MSIE supports the scripting interface and b) there exists a Object Rexx interpreter which takes advantage of these interfaces[5]. The result of this program is shown in figure 2.

---

[3] In effect, the invocation of functions (later dubbed "methods") in programs running in other process sapces (and even on other machines in a network) is carried out with a well established technique, the "remote procedure call" (RPC).

[4] Applications need to implement the Microsoft defined “automation” COM interfaces to allow for this functionality. Scriptability in this context is also meant, if using the term “OLE automation” or “ActiveX automation” or "Active scripting".

[5] In this paper IBM’s Object Rexx, cf. [W3ORX], is employed to demonstrate that it is indeed possible to use non-Microsoft programming languages for the purpose of scripting. Microsoft supplies “JScript” (JS) and “Visual Basic Script Edition” (VBScript, VBS) with some of its Windows operating systems as part of WSH, and also with the Microsoft Internet Explorer (MSIE) browser.

```
-- remote control MS Internet Explorer from Object Rexx
myIE=.OLEObject~new("InternetExplorer.Application") -- create an MSIE instance
myIE~visible=.true                                  -- make the window visible
myIE~navigate("http://www.wu-wien.ac.at")           -- navigate to the given URL
```

Figure 1: Remote Controlling (Scripting, Automating) the Microsoft Internet Explorer to Load and Display the Homepage of the Vienna University of Economics and Business Administration (Wirtschafsuniversität Wien, WU) with Object Rexx.

Practically all of Microsoft's software allows for scriptability in such a manner, e.g. ADSI (Active Directory Service Interface) for maintaining important directory information, or WMI (Windows Management Instrumentation) for managing hardware and software resources in a network.

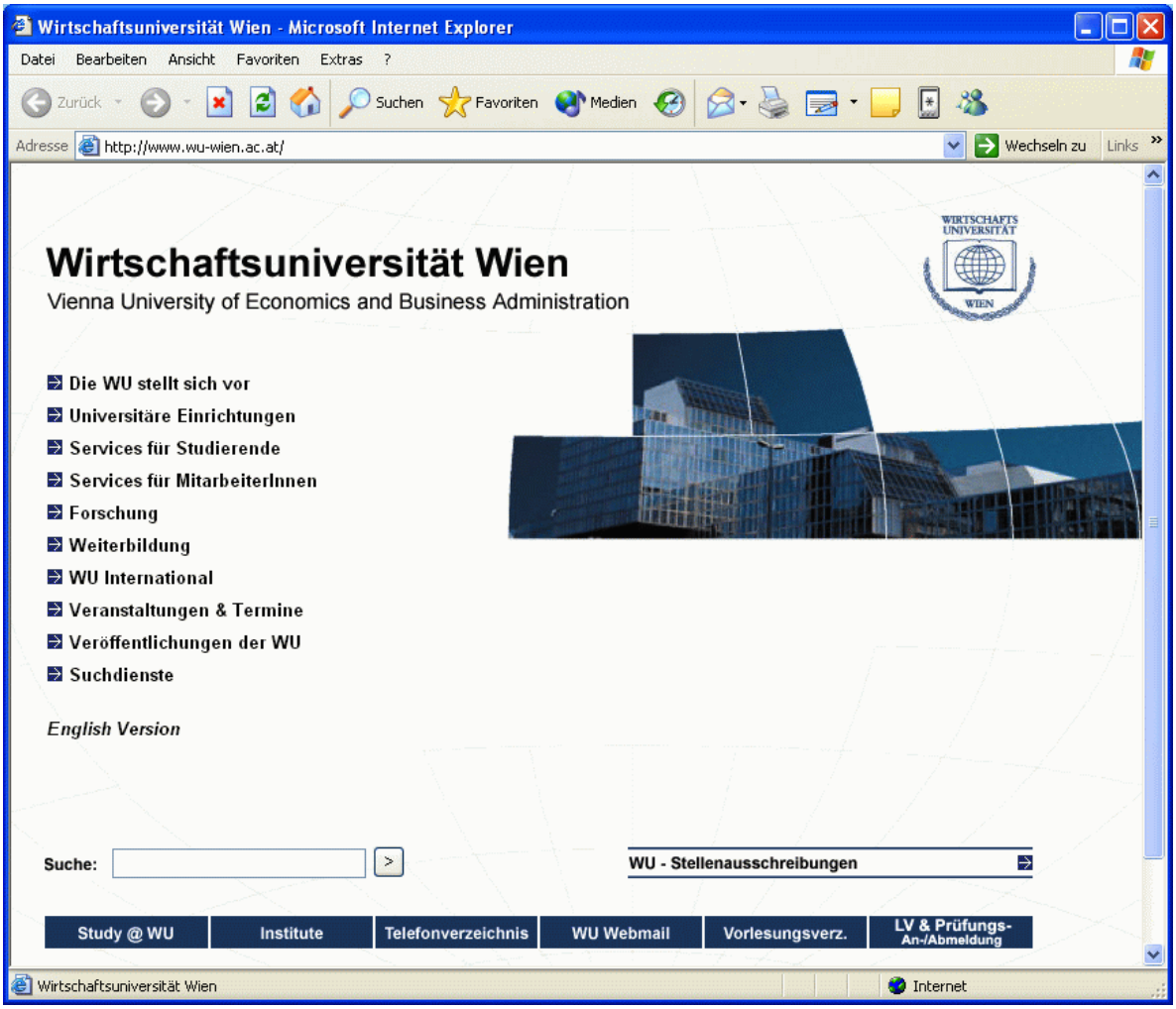

Figure 1a: Result of the Object Rexx Program of Figure 1.

2.1 Windows Script Host and Windows Script Engine

At the end of the 90'ies Microsoft enhanced these sets of scripting interfaces by allowing applications and scripting languages to interact even closer with each other, by defining a set of interface rules, which became known under the name "ActiveX Scripting" and lately as "Windows Scripting".

Firstly, any programming language which supports these interfaces[6)] in form of an OLE COM object can be used to script *any* scriptable application and is called "Windows Script Engine" (WSE). Secondly, an application knowing about these interfaces can setup an environment for the WSE scripts to work with. If such an additional work is undertaken by scriptable applications, they are called "Windows Script Hosts" (WSH). In order for the market to adapt to these technologies in an easier way, Microsoft supplies pre-fabricated COM objects which makes the implementing of these interfaces rather simple.[7)]

One specific enhancement of WSH is the ability for WSHapplications to make its objects up-front (implicitly) available to the scripts. This way the programmers of scripts do not need to explicitly get access to the objects of applications they need to remote control, thereby saving quite some coding efforts. This specific feature is of utmost importance in the realm of the Microsoft application "Internet Explorer", which makes it possible to refer to the HTML document and to its nodes right away.[8)]

Microsoft supplies the following three Windows Script Host (WSH) applications for Windows:

- "Microsoft Internet Explorer" (MSIE),
- "Internet authoring tools" in the context of the "Internet Information Server" (IIS: enabling the full scriptability for its "Active Server Pages", ASP),

---

6) Cf. [W3WSH], Microsoft Windows Script 5.6 Documentation, section "Microsoft Windows Script Interfaces / Introduction", interfaces "IActiveScript", "IActiveScriptParse" and "IPersist".

7) Eg. [W3Delphi] uses the "Microsoft Script Control" COM object to make a Delphi application scriptable in a few lines of code.

8) This specific feature is of utmost importance in the realm of the Microsoft application "Internet Explorer", which makes it possible to refer to the HTML document and to its nodes right away.

- "Shell"[9].

2.1.1 The MSIE Windows Script Host

The MSIE Windows Script Host allows Windows script engines to directly use the Microsoft implementation of DOM, which the company has been marketing with the acronym" DHTML",[10] thereby allowing scripts to interact programmitcally with the HTML- and XML-documents MSIE processes and renders.

Figure 2 depicts an Object Rexx script which creates HTML text using the DOM/DHTML interfaces supplied by MSIE, which prepares the Object Rexx environment such that it becomes possible for Object Rexx scripts to directly interact with MSIE's own objects. Figure 2a shows the rendering results of loading the HTML text of figure 2 with the Microsoft Internet Explorer.

This way it becomes possible for *any* WSE to create, alter or delete parts or all of a HTML document while the HTML document gets loaded and/or while it is presented to the end user! As such Visual Basic Script Edition (cf. figure 3, rendering in figure 3a) as well as Jscript (cf. figure 4, rendering in figure 4a) can be readily used for the very same purpose.

```
<head>
<title>Demonstrating the Object Rexx Windows Script Engine (WSE)...</title>
</head>

<body>
  <script language="Object Rexx">
     document~writeln( "Greetings from Object Rexx!" )
  </script>
</body>
```

Figure 2: An Object Rexx Program ("`orexx.html`") Creating the Content of a HTML Document.

---

9) "Shell" has been dubbed "Windows Scripting Host" (WSH) lately, because it serves as a Windows' scripting host that allows any scripting language to be used for creating and executing classic maintenance scripts for the management of Windows systems themselves, like installing applications, maintaining the registry, defining users and creating logon scripts for them and the like. Of course, this has turned "WSH" into a *very* confusing homonyme!

10) "DOM" ("document object model") is a standard of the World Wide Web consortium, cf. [W3C], for defining the application programming interfaces for programs whishing to manipulate HTML (XML) documents. "DHTML" ("dynamic HTML") is used as a synonym for "DOM" and sometimes only denotes Microsoft's implementation of DOM.

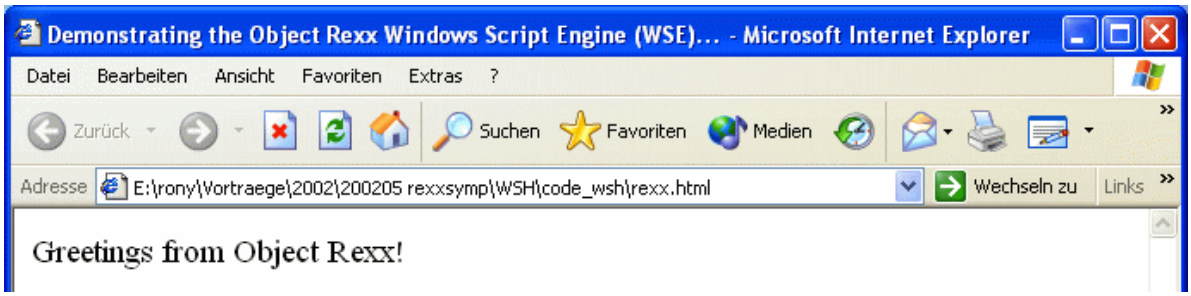

Figure 2a: Rendering of the HTML Text Containing a Script of Figure 2.

```
<head>
<title>Demonstrating the VBScript Windows Script Engine (WSE)...</title>
</head>

<body>
  <script language="VBScript">
     document.writeln "Greetings from VBScript!"
  </script>
</body>
```

Figure 3: A Visual Basic Script ("`vbs.html`") Program Creating the Content of a HTML Document.

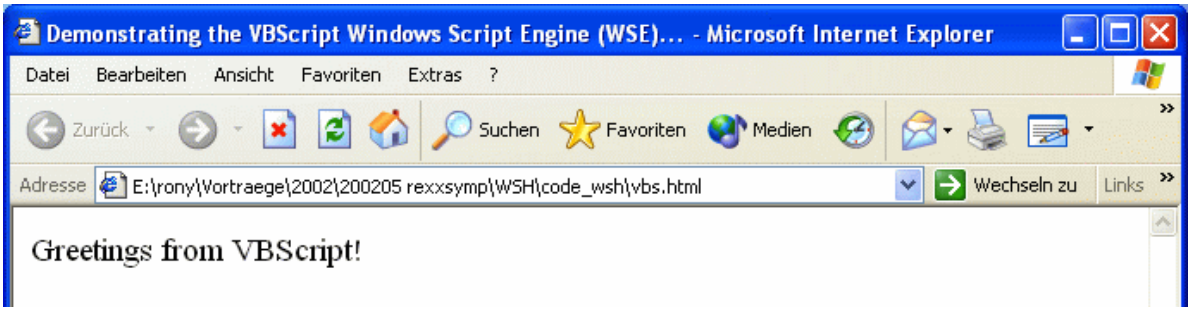

Figure 3a: Rendering of the HTML Text Containing a Script of Figure 3.

```
<head>
<title>Demonstrating the JScript Windows Script Engine (WSE)...</title>
</head>

<body>
  <script language="JScript">
     document.writeln( "Greetings from JScript!" )
  </script>
</body>
```

Figure 4: A JScript ("`js.html`") Program Creating the Content of a HTML Document.

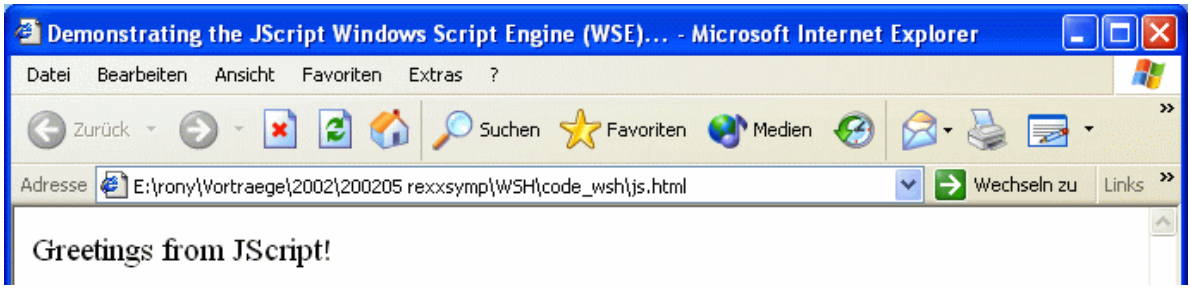

Figure 4a: Rendering of the HTML Text Containing a Script of Figure 4.

2.1.2 The “Shell” Windows Script Host

The Windows operating systems Windows 98, Windows ME, Windows 2000 and Windows XP are delivered with the aforementioned “Shell”[11)] for allowing the scripting of these operating systems. For that purpose Microsoft’s Windows scripting engines Visual Basic Script Edition (VBScript, VBS) as well as its JScript (JS) are installed on such Windows operating systems.[12)]

For helping to maintain Windows operating system installations, “Shell” supplies scriptable COM objects for making it easier to create Windows scripts, the most important one being “Wscript” which itself makes other scriptable objects available, e.g. “WshArguments” (to get and parse supplied named and unnamed arguments to the script itself), “WshController” (to invoke and control scripts, even on other Windows computers), “WshNetwork” (to maintain network resources like network

```
-- Object Rexx using the "Shell" WSH
wsn = .OLEObject~new("WScript.Network")
wscript~echo( "ComputerName:" wsn~ComputerName )
wscript~echo( "UserName:    " wsn~UserName )
wscript~echo( "UserDomain:  " wsn~UserDomain )
```

Figure 5: An Object Rexx WSE Script ("`query.rxs`"[1)]) Querying and Displaying the Computer’s Name, the Logged on User’s Windows Name and the Windows Domain the User Belongs To.

```
E:\rony\Vortraege\2002\200205 rexxsymp\WSH\code_wsh>cscript query.rxs
Microsoft (R) Windows Script Host, Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. Alle Rechte vorbehalten.

ComputerName: WIWI-DHCP-002
UserName:     Administrator
UserDomain:   WIWI-DHCP-002
```

Figure 5a: A Possible output of running the program in figure 5 from the command line ("`cscript query.rxs`").

---

11) Please be advised, that many sources (including Microsoft) are using the generic term “Windows scripting host” (WSH) to also denote the “Shell” Windows scripting host itself.

12) The “Shell” Windows scripting host can be downloaded and installed on to computers running the Windows 95 and Windows NT operating systems as well, cf. [W3WSH]. The Windows scripting host with its engines “Visual Basic Script Edition” (VBScript, VBS) and “JScript” (JS) are installed also with the MSIE Windows scripting host on Windows 95 and Windows NT systems. Updates of WSH occur usually implicitly with updates to MSIE as well on all Windows operating systems.

shares and network printers) and “WshShell” (to maintain registry entries and or Windows shortcuts, to work with the Windows “Special Folders” etc.).

Figure 5 shows an Object Rexx script which displays the name of the computer, the name of the logged on user and the domain name of the user, who runs that particular script. Figure 5a depicts a possible output, if run from a command line window.

In addition there are scriptable COM “utility” objects available, like

- a COM directory object (“Scripting.Directory”) which allows to store, retrieve and delete information (objects) with a specific, program supplied key value,
- a COM object (“Scripting.FileSystemObject”, FSO) for creating, maintaining, copying, deleting any type of streams organized as files.

2.2 Windows Script Components (WSC)

Due to the proliferation of the scripting of Windows applications, it has become interesting to create scripting components which one could use from other programming languages and scripts. In the context of defining the Windows scripting architecture, Microsoft devised the ability to define COM compliant interface definitions, where the implementation of the announced functions (methods) and attributes can be done in any Windows scripting engine (WSE)!

The devised framework is called “Windows Script Component” (WSC), where the interface descriptions[13)] and the implementation of the COM component are layed out in a plain XML file. The burden of creating at runtime instances of such Windows script components is taken away from the implementors, making the creation and usage of such a component as a result very simple. WSCs are true COM compliant components and it is even possible to distribute the functionality of such WSC over networks, by making them available via DCOM.

[13)] Technically, one can define the COM interfaces for attributes, functions (methods) and events which may get triggered from the componente.

```
<?xml version="1.0"?>
<component>

<?component error="true" debug="true"?>

<registration
        description="Counter"
        progid="Rexx.Counter"
        version="1.00"
        classid="{cfe63bb0-391f-11d6-a3d7-006094eb4d95}"
/>

<public>
        <property name="counter">
                <get/>
        </property>
        <method name="increment" />
</public>

<script language="Object Rexx">
<![CDATA[

   .local~counter=100           /* initialize counter to "100"  */

   ::routine increment public   /* increment counter            */
     .local~counter=.counter+1  /* increment counter            */
     return .counter            /* return value                 */

   ::routine get_counter public /* accessor for property        */
     return .counter            /* return value                 */

]]>
</script>

</component>
```

Figure 6: A Windows Script Component (WSC, "`Counter.wsc`") Implemented with Object Rexx, Realizing a Simple Counter.

Figure 6 depicts an Object Rexx WSC[14], which demonstrates the easiness of creating a Windows script component by defining a read-only attribute[15] "`counter`" and a function "`increment_counter`" which adds 1 to the existing value of the "`counter`" attribute.

Figure 7 depicts a VBScript program, figure 8 shows a JScript program and figure 9 shows an Object Rexx program, which use the Object Rexx Windows script component of figure 6 for counting purposes. Figure 10 shows how these three Shell scripts can be started and what output they will yield.

---

[14] This text file needs to be registered with Windows. One way to accomplish this is using the Windows Explorer, displaying the file, right-clicking upon it and choosing the menu option "register". After registering, this very COM class (realized with Object Rexx!) can be used from any programming language.

[15] Attributes can be defined to be read-only, read-/writable, write-only.

```
' VBScript
dim MyVar
Set MyVar = createObject("Rexx.Counter")

wscript.echo "Counter: " & MyVar.counter
wscript.echo "Counter: " & MyVar.increment
```

Figure 7: A VBScript (VBS) Program Using the Windows Script Component of Figure 6..

```
// JScript
var MyVar
MyVar = new ActiveXObject("Rexx.Counter")

WScript.echo( "Counter: " + MyVar.counter )
WScript.echo( "Counter: " + MyVar.increment() )
```

Figure 8: A JScript (JS) Program Using the Windows Script Component of Figure 6..

```
-- Object Rexx
MyVar = .OLEObject~new("Rexx.Counter")

wscript~echo( "Counter:" MyVar~counter )
wscript~echo( "Counter:" MyVar~increment )
```

Figure 9: An Object Rexx Program Using the Windows Script Component of Figure 6..

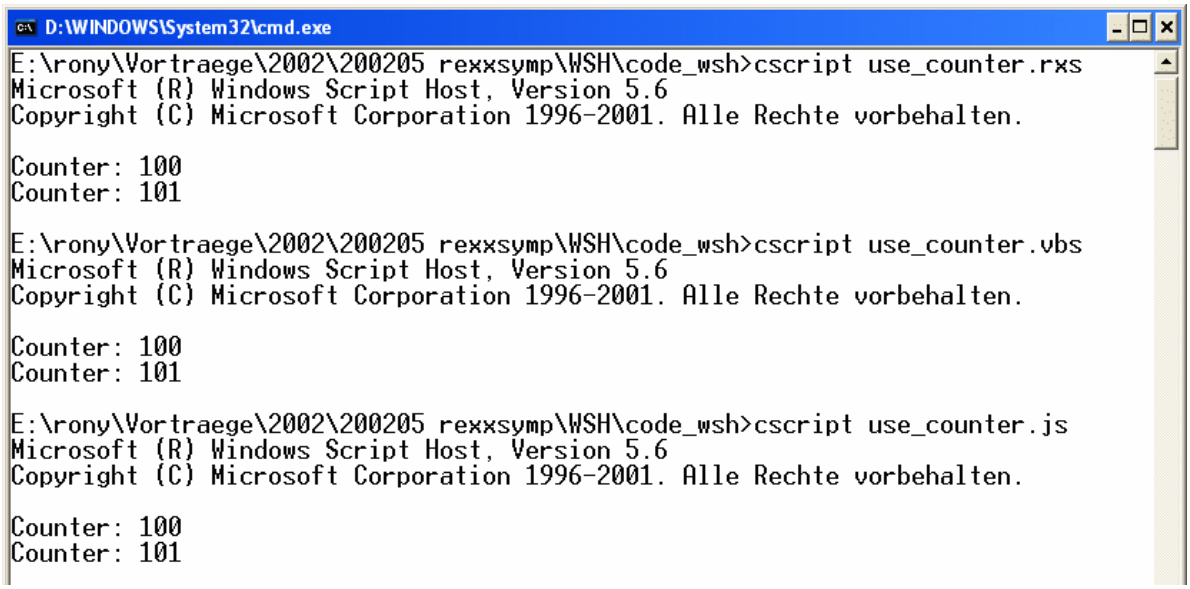

Figure 10: Invoking and Output of the Shell Scripts of Figures 7, 8 and 9.

2.3 Security Considerations

In principle, scripts execute in the context of the application process on behalf of the user. Therefeore a script has all the same rights as the application for it is running, and the rights as a machine administrator on Windows 95, 98 and ME.

This may impose important security threats as because of this it becomes possible, even for end users to use scripts e.g. for

- spying around a machine, even an entire network, if shares are available,
- changing the content of a machine (in a network), i.e. settings, but also data in files and databases, or
- creating viruses[16)].

Although this huge security problem has been known for a long time, Microsoft only started to actively pursue a more secured execution of Windows scripts by introducing a signature mechanism which - like with ActiveX controls - allows to determine with a public key mechanism, whether a script was tampered with, since it got signed. The idea behind this scheme is, that an administrator defines trustworthy signers (e.g. inhouse programmers) of scripts whose programs are to be trusted and therefore can be run.[17)]

On Windows 2000 or XP machines one needs to employ the “local security policy” mechanism to define which files with what signatures may be executed and which may not.

On all other systems, a registry key can be set such, that only three conditions are expressable: run all scripts (trust all scripts), in the case of an untrusted script prompt the user what to do or just run trusted scripts only.

---

16) In effect, the famous “Love Letter Virus” and all of its descendants are nothing else but viruses written as Windows scripts (using VBScript as the programming language!), which remote control Outlook and use the user’s address book to re-send it to all found adressees in the entire world. For that reason anti virus software like Norton’s AntiVirus, cf. [W3NAV], offer shelter by blocking Windows scripts received either via HTML documents or Outlook and the like, including the ability to check the outgoing e-Mail for virus in its attachments.

17) This does not inhibit the discussed security threats per se, as trusted people and trusted companies may be malign in effect

In order to determine whether a script is trustable, that script needs to be signed with a private key. With the corresponding public key it becomes possible to determine whether a script was tampered since it got signed. Only with version 5.6 of the Windows scripting host distribution has it become possible to sign scripts themselves in the fall of 2001, which is a pre-requisite for employing the trust-concept.[18)]

Comparing the security measurements available to Windows script with those available to Java, the concept of an authorative security manager[19)] is missing, which would be able to note and intercept actions from programs, which the policy of a company does not want to allow. One application of the elaborate Java security manager can be seen with the “sandbox policy” for Java applets, which as a result are not able to access the resources of their host machines, no matter how acquainted programmers are with Java.

In the context of Windows scripting one could employ this very same concept of security managers, if the scripting languages themselves support one, which is not the case with the Microsoft supplied scripting languages VBScript or JScript. In the context of this paper the Object Rexx WSE[20)] possesses a security manager, which would allow for creating a sandbox-equivalent environment, at least for executing Object Rexx scripts.

---

18) Although now scripts can be trusted, this does not insulate Windows users of malicious scripts by any means There have been incidents reported where trustworthy ActiveX components were still created with a malicious intent. The same can be thought of within companies where trustworthy programmers create scripts, whith malicious intents (e.g. because they got disciplined or are angry with their employer for one or another reason).

19) A security manager is part of the runtime environment and closely controls the execution of code. Using these features it becomes possible to determine which actions are allowed under which conditions and to implement security measurements, which effectively prohibit the execution of untrusted segments of code.

20) To be precise: IBM’s Object Rexx possesses a built-in security manager. Using it one can easily create a Java-like sandbox environment with it, making it even possible to execute untrusted code withouth risking harm to systems or the environment under which such programs execute.

3 SUMMARY AND OUTLOOK

This article introduced the reader to the architecture of the Microsoft "Windows Script Host" architecture which allows any programming language to be employed as a scripting language. Starting out with OLE automation, Windows application have become automatable, i.e. scriptable. The latest developments have been covered with this article, stressing the possibilities of this technology.

As scripting languages in the Windows environment usually possess a simple syntax, it becomes possible even for end-users to employ them. This way recurrent tasks can be automated and any Windows application which supports it, can be remote controlled. Complex, but complete automation solutions can be re-deployed by turning them into Windows script components (WSC) addressable from any COM aware program, including other WSE scripts.

Although the security of applying scripts needs to be tackled in a more professional way, businesses and organiziations can devise and employ organizational measurements to make it safer to use scripts. For this purpose the abiltiy to sign scripts, as introduced in the fall of 2001 with WSH 5.6, allows to define and determine at least the measure of "trust". For more secure applications one needs to refer to script programming languages which implement a bullet proof security manager like Object Rexx, which in turn can be used to create a sandboxed, i.e. secure execution environment.

4 REFERENCES

[W3C] Homepage of the World Wide Web Consortium, URL (2002-06-15):
http://www.w3.org/

[W3Delphi] Groves M.: “Scripting your Delphi Applications”, URL (2002-06-15):
http://www.madrigal.com.au/papers/scripting/scripting.htm

[W3NAV] Norton’s AntiVirus (Symantec) Virus Information Database, URL (2002-06-15):
http://securityresponse.symantec.com/avcenter/vinfodb.html/

[W3ORX] Object Rexx homepage of IBM, URL (2002-06-15):
http://www.ibm.com/software/ad/obj-rexx/

[W3WSH] Microsoft’s scripting homepage, URL (2002-06-15):
http://msdn.microsoft.com/scripting

Date of Article: 2002-06-15.

Published in: Proceedings of the „2002 International Rexx Symposium“, April 29th - May 1th, Raleigh, NC, USA 2002.

Presented at: „The 2002 International Rexx Symposium“, April 29th - May 1th, Raleigh, NC, USA 2002.