Abteilung für Wirtschaftsinformatik

An Introduction to Procedural and Object-oriented Programming (Object Rexx) 8

"Automating Windows and Windows Applications"

➔ Windows Script Host (WSH), Windows Script Engine (WSE), Windows Script File (WSF), Windows Script Component (WSC)

➔ Object Rexx vs. MS Visual Basic Script ("VBScript")

➔ Windows Configuration of File Types and their Associated Applications

Prof. Dr. Rony G. Flatscher

Windows Script Host (WSH), Overview 1

- Windows Script Host (WSH)
  - Advancement of OLE (Automation)
  - Made available at the end of the 90's
  - Part of the Windows operating system since
    - Windows 98 (16-Bit Windows)
    - Windows 2000 (32-Bit Windows)
  - Installed with Microsoft Internet Explorer (MSIE)
    - Updates via MSIE
  - Since October 2001 available also as a separate package
    - WSH 5.6 with Documentation and its wizard, search
      http://social.msdn.microsoft.com/Search/en-us/?query=download+wsh+5.6
      http://social.msdn.microsoft.com/Search/en-US/?query=download%20wsh%20wizard&ac=8

Windows Script Host (WSH), Overview 2

- Windows Script Host (WSH)
  - *Each* Application that uses the IActiveScript COM interfaces
  - WSH-Applications are able to pre-register scriptable objects with the runtime of the WSE, *before* the script program is invoked
    - No cumbersome initializing of needed objects necessary anymore!
    - *Attention of name collisions/name clashes !*
- Windows Script Host Applications of Microsoft
  - Internet Explorer (MSIE)
  - Internet Authoring Tools (IIS, Internet Information Server)
    - Active Server Pages (ASP)
  - Shell
    - Eventually new synonym (homonym!): *"Windows Script Host"* ***!!!***

Windows Script Host Engine (WSE), Overview

- Each programming language which supports the following COM interfaces
  - ActiveX/OLE Automation interfaces and
  - IActiveScript, IActiveScriptParse, IPersist
- Application can pre-register scriptable objects for the scripts
- Microsoft WSE as part of WSH
  - Visual Basic Script Edition: "VBScript"
  - Java Script ("ECMA"-Script) Microsoft Edition: "JScript"
- ooRexx on Windows is implemented as a WSE !

Windows Script File (WSF), Overview

- Collection ("package") of
  - Tasks ("job"), which may consist of different
  - Scripts
    - May be implemented in any WSE
    - WSE scripting languages can be intermixed!
- WSF files are structured with XML-Markup
  - Code must be defined as a "CDATA" section!

```
<package>
   <job ...>
      <script ...>
          <![CDATA[ ... Programmcode ...
          ]]>
      </script>
   </job>
</package>
```

Windows Script Component (WSC), Overview

- Collection of functions (methods), attributes and events, which may be implemented in any WSE
  - WSE languages can be intermixed!
- WSC files are structured with XML markup
- Definitions are registered as a COM class with Windows and are made systemwide available
  - All programs are able to use such WSC programs
  - With "Shell" WSC can be addressed via DCOM
    - WSC are therefore distributable via networks !

WSH, Example: MSIE ("DHTML"), 1

- WWW browser parses files with markup
  - HTML
  - XML
- Enables each WSE to be used for scripts
  - *Implicitly* supplied scriptable DOM objects, e.g.
    - Object "window"
    - Object "document" with all nodes
  - DOM events, e.g.
    - Keyboard, mouse, session information
- Microsofts name for DOM: DHTML
  - "Dynamic HTML"

WSH, Example: MSIE ("DHTML"), 2

```
<head>
   <title>Demonstrating the REXX Windows
          Scripting Engine (WSE)...</title>
</head>
<body>
   <script language="Object Rexx">
      document~writeln( "Greetings from REXX!" )
   </script>
</body>
```

WSH, Example: MSIE ("DHTML"), 3

```
<head>
   <title>Demonstrating the VBScript Windows
          Script Engine (WSE)...</title>
</head>
<body>
   <script language="VBScript">
      document.writeln "Greetings from VBScript!"
   </script>
</body>
```

WSH, Example: MSIE ("DHTML"), 4

```
<head>
   <title>Demonstrating the JScript Windows
          Script Engine (WSE)...</title>
</head>
<body>
   <script language="JScript">
      document.writeln( "Greetings from JScript!" )
   </script>
</body>
```

WSH, Example: "Shell", 1

- Enables the interaction with the Windows user interface
  - Implicit scriptable object named "WScript"
    - Input, output of information
    - Parsing arguments
    - Maintaining the graphical user interface
    - Access to the Windows registry
    - Installation of Windows applications
    - Network settings
    - ...
- Supplies among other things the following ActiveX/OLE-Automation classes
  - "Scripting.Directory"
  - "Scripting.FileSystemObject" (FSO)

WSH, Example: "Shell", 2

- Starting via the commandline interface (CLI)
  - To get a CLI
    - enter "Start → Run ..." the command "`cmd.exe`"
    - Invocation of "Shell" script programs

      `wscript scriptProgramName`
      - Output via alert popup windows
    - or

      `cscript scriptProgramName`
      - Output to the CLI
- Double-click using the Explorer, if files end in
  - **`.vbs`** (VBScript)
  - **`.js`** (JScript)
  - **`.rxs`** (Object Rexx)

WSH, Example: "Shell", 3

- Query some information about the Windows computer

```
/* "query.rxs": 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 )
```

WSF, Example, 1 "getVersion.wsf"

```
<?xml version="1.0"?>
<?job error="true"?>   <!-- uebeprueft, ob XML korrekt ist -->

<package id="rgf_version">

  <!-- Allererster "job" ist "Default"-job -->
  <!-- "cscript|wscript getVersion.wsf"    -->
  <!-- "cscript|wscript getVersion.wsf //job:firstJob" -->
  <job id="firstJob">
     <script language="JScript"><![CDATA[
           WScript.echo( "Hi, this is JScript." );
        ]]>
     </script>

     <script language="Object Rexx"><![CDATA[
              wscript~echo( "Hi, this is Object Rexx." )
        ]]>
     </script>

     <script language="VBScript"><![CDATA[
              wscript.echo "Hi, this is VBScript."
        ]]>
     </script>
  </job>
```

```
  <!-- "cscript|wscript getVersion.wsf //job:secondJob" -->
  <job id="secondJob">
     <script language="JScript"><![CDATA[
          function jsVersion(){
            return ScriptEngine() + ": " + ScriptEngineMajorVersion() +
                   "." + ScriptEngineMinorVersion() + " build: " +
                   ScriptEngineBuildVersion();
          }
        ]]>
     </script>

     <script language="VBScript"><![CDATA[
          function vbsVersion()
             vbsVersion=ScriptEngine() & ": " & ScriptEngineMajorVersion() _
                        & "." & ScriptEngineMinorVersion() & " build: " & _
                        ScriptEngineBuildVersion()
        end function
        ]]>
     </script>

     <script language="Object Rexx"><![CDATA[
          ::routine rxsVersion public
            return ScriptEngine()":" ScriptEngineMajorVersion()"." || -
                   ScriptEngineMinorVersion() "build:" –
                   ScriptEngineBuildVersion()
        ]]>
     </script>

     <script language="Object Rexx"><![CDATA[
          wscript~echo( "(Rexx) VBScript tells me:" vbsVersion()"." )
          wscript~echo( "(Rexx) JScript tells me: " jsVersion()"." )
          wscript~echo( "(Rexx) Rexx tells me:    " rxsVersion()"." )
        ]]>
     </script>
  </job>
</package>
```

WSF, Example, 2

- Activate
  - Double-click in the Windows explorer
    - Invokes the "default job" ("firstJob")
  - CLI
    - Invokes the "default job", always 1st job ("firstJob")

```
cscript getVersion.wsf
wscript getVersion.wsf
```

    - Invoke job which is named "firstJob"

```
cscript getVersion.wsf //job:firstJob
wscript getVersion.wsf //job:firstJob
```

    - Invoke job which is named "secondJob"

```
cscript getVersion.wsf //job:secondJob
wscript getVersion.wsf //job:secondJob
```

WSC, Example, 1 "Counter.wsc"

```
<?xml version="1.0"?>
<?component error="true" debug="true"?>
<component>
    <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 function
          return .counter                   -- return value
      ]]>
    </script>
</component>
```

WSC, Example, 2

- WSC definitions must be registered with Windows
  - Windows Explorer
  - Right mouse-click over WSC file
    - Choose "Register"
    - Choose "Generate Type Library"
- Usable as any other ActiveX/OCX/OLE COM classes
  - Can be also analyzed via the utility "rgf_oleinfo.hta"

WSC, Example, 3

```
' VBScript: "use_counter.vbs"
dim MyVar
Set MyVar = createObject("Rexx.Counter")
wscript.echo "Counter: " & MyVar.counter
wscript.echo "Counter: " & MyVar.increment
```

```
// JScript: "use_counter.js"
var MyVar
MyVar = new ActiveXObject("Rexx.Counter")
WScript.echo( "Counter: " + MyVar.counter )
WScript.echo( "Counter: " + MyVar.increment() )
```

```
-- REXX: "use_counter.rxs"
MyVar = .OLEObject~new("Rexx.Counter")
wscript~echo( "Counter:" MyVar~counter )
wscript~echo( "Counter:" MyVar~increment )
```

WSH, Security Considerations, 1

- Script programs run under the context (authority) of the program, that started it
  - Access
    - Locally ("standalone PC"?)
    - Network
  - Spying
  - Sabotage
    - Changing/deleting of content
  - Creation of viruses
    - Original "Love Letter Virus" and MS Outlook
  - Attention also at all of the functionality the WSH gives its script programs!

WSH, Security Considerations, 2

- Systematic security measurements rather late (Oktober 2001)
  - WSH 5.6
    - "Signing" of script programs
      - Using the concept of "trust"
    - Possibility to set the execution rights based on trust
- Modest security measurement
  - E.g. problem of frustrated/dishonest personnel whose script programs are marked "trusted"
- No "sandbox" for WSH!
  - Therefore use the security manager of the script languages you are using, if available at all
    - E.g. ooRexx' security manager

ooRexx vs. Visual Basic (Script Edition), 1

ooRexx

- Message operator

  ~ (Tilde)
- Continuation character

  , (comma) or – (dash)
- String concatenation

  (space between strings)

  || (2 vertical bars)
- Defining variables

  Just denote the name

VBScript

- "Message" (dereference) operator

  . (dot)
- Continuation character

  _ (underline)
- String concatenation

  & (ampersand)
- Defining variables

  DIM var_names

ooRexx vs. Visual Basic (Script Edition), 2

ooRexx

- Line comment
  - `--` (2 dashes)
- Multi-line comments
  - `/* ... */`
    - May span multiple lines
    - May be nested

VBScript

- Line comment
  - `'` (apostroph)
  - REM
    - Abbreviation for REMark
    - No statement before it allowed
  - `:` REM
    - If following a statement in the same line, *must* be preceeded by a column surrounded by a space

ooRexx vs. Visual Basic (Script Edition), 3

ooRexx

- Calling a procedure

```
CALL proc1 a1, a2, a3
```

- Calling a function

```
a=proc1(a1, a2, a3)
```

or:

```
CALL proc1 a1, a2, a3
a=result
```

VBScript

- Calling a procedure

```
CALL proc1(a1, a2, a3)
```

or:

```
proc1 a1, a2, a3
```

- Calling a function

```
a=proc1(a1, a2, a3)
```

ooRexx vs. Visual Basic (Script Edition), 4

ooRexx

- Calling a function

```
a=proc1( , , a3)
```

oder:

```
CALL proc1 , , a3
a=result
```

VBScript

- Calling a function

```
a=proc1(a1, a2, a3)
```

- Calling a function using named arguments, e.g.

```
a=proc1( a3 := "Das 3. Argument!" )
```

ooRexx vs. Visual Basic (Script Edition), 5

ooRexx

- Defining a procedure

```
proc1: procedure
  use arg a1, a2, a3
  say "a1="a1 "a2="a2 "a3="a3
  return
```

or:

```
::routine proc1
  use arg a1, a2, a3
  say "a1="a1 "a2="a2 "a3="a3
```

- Defining a function

```
proc1: procedure
  use arg a1, a2, a3
  return a1 || a2 || a3
```

or:

```
::routine proc1
  use arg a1, a2, a3
  return a1 || a2 || a3
```

VBScript

- Defining a procedure

```
Sub proc1(a1, a2, a3)
   MsgBox "a1=" & a1 & " a2=" & a2_
          " a3=" & a3
End Sub
```

- Defining a function

```
Func proc1(a1, a2, a3)
   proc1=a1 & a2 & a3
End Func
```

ooRexx vs. Visual Basic (Script Edition), 6

ooRexx

- Can be represented as

```
MyLabel~Height  = 2000
MyLabel~Width   = 2000
MyLabel~Caption = "This is MyLabel"
```

- or

```
m=MyLabel
m~Height  = 2000
m~Width   = 2000
m~Caption = "This is MyLabel"
```

VBScript

- WITH statement

```
With MyLabel
   .Height  = 2000
   .Width   = 2000
   .Caption = "This is MyLabel"
End With
```

- without WITH statement

```
MyLabel.Height  = 2000
MyLabel.Width   = 2000
MyLabel.Caption = "This is MyLabel"
```

ooRexx vs. Visual Basic (Script Edition), 7

```
' VBScript: "use_counter.vbs"
dim MyVar
Set MyVar = createObject("Rexx.Counter")
wscript.echo "Counter: " & MyVar.counter
call wscript.echo( "Counter: " & MyVar.increment )
```

```
-- REXX: "use_counter.rxs"
MyVar = .OLEObject~new("Rexx.Counter")
wscript~echo( "Counter:" MyVar~counter )
wscript~echo( "Counter:" MyVar~increment )
```

Windows Configuration of File Types and their Associated Applications, 1

- Installation programs, 1
  - Associate file types with identifiers
    - CLI command "assoc"

      `assoc /?`

      - Explains the command

      `assoc`

      - Lists all currently defined associations

      `assoc | more`

      - Lists all currently defined associations, stops output if CLI window is full

      `assoc .rex`

      - Lists the association of the given file type

      `assoc .recks=rexxfile`

      - Defines a new association: files with the file type "**`.recks`**" get associated with the identifier "**`rexxfile`**"

Windows Configuration of File Types and their Associated Applications, 2

- Installation programs, 2
  - Define default programs to execute associated file types
    - CLI command "ftype"

`ftype /?`

- Explains the command

`ftype`

- List all currently defined default programs to execute associated file types

`ftype | more`

- List all currently defined default programs to execute associated file types , stops output if CLI window is full

`ftype rexxfile`

- Lists the defined default program with all pre-set arguments for the given associated file type

`ftype rexxfile=D:\Programme\ooRexx\rexx.exe "%1" %*`

- Defines the default program ("rexx.exe") which executes the associated file type (associate file type with the identifier "rexxfile") and defines the pre-set arguments for starting the program

Windows Configuration of File Types and their Associated Applications, 3

- Loading/executing of files with their default programs
  - Windows Explorer
    - Double-click with the mouse on the file
    - Select the file and press the "enter" key on the keyboard
    - Click right mouse button and select "Open"
  - CLI
    - Enter the file name *with* its file type and press the "enter" key

Windows Configuration of File Types and their Associated Applications, 4

- Loading/executing of files with their default programs
  - CLI (continued)
    - File type may be omitted, *if* the environment variable "**PATHEXT**" is defined
      - List the actual values of the environment variable in the CLI window

```
echo %pathext%
```

Output (maybe) :

```
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.REX
```

      - Adding a new value to the environment variable "PATHEXT"

```
set PATHEXT=%pathext%;.RECKS
```

      - Define it for all Windows sessions
        - Select "Properties → Extended → Environment Variables" of the context menu for the object "My Computer"
        - Pick "System Variables" in the lower part of the window and locate the entry "PATHEXT", add the file type and save the changes

Windows Configuration of File Types and their Associated Applications, 5

- Concluding example
  - Define a proper file type for ooRexx programs

```
assoc .recks=recksFiles
ftype recksFiles=D:\Programme\ooRexx\rexx.exe "%1" %*
set   pathext=%pathext%;.RECKS
```

  - From now on all files with the file type ".recks" will be executed by ooRexx (= default program for "**recksFiles**"), e.g. entering the following in a CLI:

```
mySuperDuper_ooRexxProgram.recks
```

  - In this case it is not necessary to supply the file type for ".recks" files, e.g. entering the following in a CLI:

```
mySuperDuper_ooRexxProgram
```

Assignments and Presentations

- Group assignment for the next (final) date
  - One WSF file with three (short!) contained jobs, which each should consist of at least two scripts
  - One WSC file, which will be used from a DHTML program
  - Automation ("Remote-controlling") of Windows programs
    - Combine at least three Windows programs
    - At least one of these Windows programs must not be from Microsoft!
  - Presentation of the examples
    - At most twenty minutes per group, discussion/Q&A time: five minutes
    - Professional slides (e.g. in OpenOffice)
    - *Each* group member *must* present!