Institut für Betriebswirtschaftslehre und Wirtschaftsinformatik

Abteilung für Wirtschaftsinformatik

Automatisierung von Windows Anwendungen (8)

➔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 Konfiguration von Dateien und ihren Anwendungen

Prof. Dr. Rony G. Flatscher

Windows Script Host (WSH), Überblick 1

- Windows Script Host (WSH)
  - Weiterentwicklung von ActiveX/OLE Automation
  - Ende der 90er verfügbar gemacht
  - Bestandteil der Windows Betriebssysteme seit
    - Windows 98 (16-Bit Windows)
    - Windows 2000 (32-Bit Windows)
  - Wird mit dem Microsoft Internet Explorer mitinstalliert
    - Updates erfolgen indirekt über MSIE
  - Als eigenständiges Paket seit Oktober 2001 downloadbar
    - WSH 5.6 mit Dokumentation

      http://msdn.microsoft.com/library/default.asp?url=/downloads/list/webdev.asp

Windows Script Host (WSH), Überblick 2

- Windows Script Host (WSH)
  - *Jede* Anwendung, die die IActiveScript-COM-Schnittstellen benutzt
  - WSH-Anwendungen sind in der Lage, automatisierbare Objekte in die Laufzeitumgebung der WSE einzubringen, *ehe* die Abarbeitung der Skriptprogramme beginn
    - Kein aufwändiges Initialisieren von benötigten Objekten mehr notwendig!
    - *Achtung auf Namenskollisionen (Englisch: "name clashes") !*
- Windows Script Hosts der Firma Microsoft
  - Internet Explorer (MSIE)
  - Internet Authoring Tools (IIS, Internet Information Server)
    - Active Server Pages (ASP)
  - Shell
    - Neues Synonym: *"Windows Script Host"* ***!!!***

Windows Script Host Engine (WSE), Überblick

- Jede Programmiersprache, die die folgenden COM-Schnittstellen unterstützt
  - ActiveX/OLE Automation Schnittstellen sowie
  - IActiveScript, IActiveScriptParse, IPersist
- Anwendungen können automatisierbare Objekte den Skriptprogrammen vorab zur Verfügung stellen
- Von Microsoft im WSH enthaltene WSE
  - Visual Basic Script Edition: "VBScript"
  - Java Script (ECMA-Script) Microsoft Edition: "JScript"
- Ursprünglich von IBM erstellte WSE: Object Rexx

Windows Script File (WSF), Überblick

- Sammlung (Paket, Englisch: "package") von
  - Aufgaben (Englisch: "job"), die aus verschiedenen
  - Skriptprogrammen (Englisch: "script") bestehen können
    - Können in jeder beliebigen WSE codiert werden
    - WSE-Sprachen können auch gemischt werden!
- WSF-Dateien werden mit XML-Markup strukturiert
  - Programmcode muss als "CDATA"-Abschnitt definiert werden!

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

Windows Script Component (WSC), Überblick

- Satz von Funktionen, Attributen und Ereignissen, die in einer beliebigen WSE-Sprache codiert werden können
  - WSE-Sprachen können auch gemischt werden!
- WSC-Dateien werden mit XML-Markup strukturiert
- Definitionen werden als COM-Objekt in Windows registriert und damit systemweit zur Verfügung gestellt
  - Alle Programme können derartige WSC benutzen
  - Mit "Shell" können WSC auch über DCOM angesprochen werden
    - WSC sind daher auch über Netzwerke verteilbar !

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

- WWW Browser zerlegt Dateien mit Auszeichnungen (Englisch: "Markup")
  - HTML
  - XML
- Erlaubt jeder WSE-Sprache volle Interaktion, z.B.
  - *Implizit* zur Verfügung gestellte DOM-Objekte, z.B.
    - Objekt "window"
    - Objekt "document" mit allen Knoten (Englisch: "nodes")
  - DOM-Ereignisse, z.B.
    - Tastatur, Maus, Sitzungsinformationen (Englisch "session infos")
- Microsofts Name for DOM: DHTML
  - "Dynamic HTML"

WSH, Beispiel: 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, Beispiel: 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, Beispiel: 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, Beispiel: "Shell", 1

- Ermöglicht die Interaktion mit der Windows Benutzerschnittstelle
  - Implizites Objekt "WScript"
    - Eingabe, Ausgabe von Informationen
    - Zerlegen von Argumenten
    - Verwaltung der Benutzeroberfläche
    - Zugriff auf die Windows Registrierungsdatenbank
    - Installation von Windows Anwendungen
    - Netzwerkeinstellungen
    - ...
- Stellt u.a. folgende ActiveX/OLE-Automation Klassen zur Verfügung
  - "Scripting.Directory"
  - "Scripting.FileSystemObject" (FSO)

WSH, Beispiel: "Shell", 2

- Kommandozeilenaufruf
  - Kommandozeilenfenster erhalten Sie, indem Sie
    - unter "Start → Ausführen ..." das Kommando "`cmd.exe`"
    - Aufruf von "Shell"-Skriptprogrammen

      `wscript Skriptdatei`
      - Ausgaben erfolgen über Popup-Fenster
    - oder

      `cscript Skriptdatei`
      - Ausgaben erfolgen über das Kommandozeilenfenster
- Doppelklick im Explorer, wenn Dateiendungen
  - **`.vbs`** (VBScript)
  - **`.js`** (JScript)
  - **`.rxs`** (Object Rexx)

WSH, Beispiel: "Shell", 3

- Einige Informationen über Windows Rechner abfragen

```
/* "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, Beispiel, 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, Beispiel, 2

- Aktivieren
  - Doppelklicken im Windows Explorer
    - Aufruf der "Default-Aufgabe" ("firstJob")
  - Kommandozeile
    - Aufruf der "Default-Aufgabe", immer 1. Aufgabe ("firstJob")
      ```
      cscript getVersion.wsf
      wscript getVersion.wsf
      ```
    - Aufruf der Aufgabe, die "firstJob" heißt
      ```
      cscript getVersion.wsf //job:firstJob
      wscript getVersion.wsf //job:firstJob
      ```
    - Aufruf der Aufgabe, die "secondJob" heißt
      ```
      cscript getVersion.wsf //job:secondJob
      wscript getVersion.wsf //job:secondJob
      ```

WSC, Beispiel, 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, Beispiel, 2

- WSC Definitionen müssen registriert werden
  - Windows Explorer
  - Rechte Maustaste über WSC-Datei
    - "Registrieren" auswählen
    - "Typbibliothek generieren" auswählen
- Benutzung wie normale ActiveX/OCX/OLE-Automatisierungsobjekte
  - Können auch über "rgf_oleinfo.hta" analysiert werden

WSC, Beispiel, 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, Sicherheitsüberlegungen, 1

- Skriptprogramme laufen im "Kontext" der Anwendung, die sie gestartet haben
  - Zugriff
    - Lokal ("standalone PC"?)
    - Über das Netzwerk
  - Spionieren
  - Sabotieren
    - Ändern/Löschen von Inhalten
  - Erzeugen von Viren
    - "Love Letter Virus" und MS Outlook
  - Achtung auch auf die gesamte Funktionalität der Wirtsapplikation, die den Skriptprogrammen zur Verfügung steht!

WSH, Sicherheitsüberlegungen, 2

- Erst sehr spät (Oktober 2001) wurden für WSH Sicherheitsmaßnahmen systematisch vorgesehen
  - WSH 5.6
    - "Unterzeichnen" (Englisch: "Signing") von Skriptprogrammen
      - Benutzung des Konzepts "Vertrauen"
    - Möglichkeit, Ausführungsrechte auf Basis von Vertrauen festzulegen
- Bescheidene Sicherheitsmaßnahme
  - Z.B. Problem von unzufriedenen/unehrlichen Mitarbeitern, deren Skriptprogramme als "vertrauenswürdig" markiert sind
- Kein "Sandkasten" (Englisch: "sandbox") für WSH absehbar!
  - Benutzen Sie daher die Security Manager jener Skriptsprachen, die Sie einsetzen
    - z.B.. IBM's Object Rexx bzw. ooRexx

Object Rexx vs. Visual Basic (Script Edition), 1

Object Rexx

- Nachrichtenoperator

  ~ (Tilde)

- Zeilenfortsetzungszeichen

  , (Beistrich) oder – (Bindestrich)

- Zeichenkettenverknüpfung

  (Leerzeichen zwischen Strings)

  || (2 senkrechte Striche)

- Variablen definieren

  einfach Namen angeben

VBScript

- Nachrichtenoperator

  . (Punkt)

- Zeilenfortsetzungszeichen

  _ (Unterstrich)

- Zeichenkettenverknüpfung

  & (Kaufmännisches "Und")

- Variablen definieren

  DIM var_namen

Object Rexx vs. Visual Basic (Script Edition), 2

Object Rexx

- Zeilenkommentar beginnt mit
  - `--` (2 Bindestriche)
- Mehrzeilige Kommentare

  `/* ... */`
  - Dürfen sich über mehrere Zeilen erstrecken
  - Dürfen ineinander verschachtelt werden

VBScript

- Zeilenkommentar beginnt mit
  - `'` (Apostroph)

    REM
    - Abkürzung für REMark (Englisch für: "Anmerkung")
    - Wenn vorher keine Anweisung

    `:` REM
    - Wenn vorher eine Anweisung steht, *muss* ein Doppelpunkt mit führenden und nachfolgenden Leerzeichen vor "REM" stehen

Object Rexx vs. Visual Basic (Script Edition), 3

Object Rexx

- Prozedur aufrufen

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

- Funktion aufrufen

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

oder:

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

VBScript

- Prozedur aufrufen

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

oder:

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

- Funktion aufrufen

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

Object Rexx vs. Visual Basic (Script Edition), 4

Object Rexx

- Funktion aufrufen

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

oder:

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

VBScript

- Funktion aufrufen

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

- Aufruf mit benannten Argumenten, z.B.

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

Object Rexx vs. Visual Basic (Script Edition), 5

Object Rexx

- Prozedur definieren

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

oder:

```
::routine proc1
  say "a1="arg(1) "a2="arg(2) –
      "a3="arg(3)
```

- Funktion definieren

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

oder:

```
::routine proc1
  return arg(1) || arg(2) || arg(3)
```

VBScript

- Prozedur definieren

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

- Funktion definieren

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

Object Rexx vs. Visual Basic (Script Edition), 6

Object Rexx

- Repräsentierbar als

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

- oder auch:

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

VBScript

- WITH-Anweisung

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

- ohne WITH-Anweisung

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

Object Rexx 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 )
```

Konfiguration von Dateien und ihren Anwendungen unter Windows, 1

- Installationsprogramme, 1
  - Assoziieren Dateitypen mit Bezeichnern
    - Kommandozeilenbefehl "assoc"

`assoc /?`

- Erläutert den Befehl

`assoc`

- Listet alle aktuell definierten Assoziationen auf

`assoc | more`

- Listet alle aktuell definierten Assoziationen auf, hält die Ausgabe an, wenn das Kommandozeilenfenster voll ist

`assoc .rex`

- Zeigt die Assoziation der Dateiendung an

`assoc .recks=rexxfile`

- Definiert eine neue Assoziation: Dateien mit der Dateiendung "**`.recks`**" werden mit dem Bezeichner "**`rexxfile`**" assoziiert

Konfiguration von Dateien und ihren Anwendungen unter Windows, 2

- Installationsprogramme, 2
  - Definieren Defaultprogramme für assoziierte Dateitypen
    - Kommandozeilenbefehl "ftype"

`ftype /?`

- Erläutert den Befehl

`ftype`

- Listet alle aktuell definierten Defaultprogramme für assoziierte Dateitypen auf

`ftype | more`

- Listet alle aktuell definierten Defaultprogramme für assoziierte Dateitypen auf, hält die Ausgabe an, wenn das Kommandozeilenfenster voll ist

`ftype rexxfile`

- Zeigt die Kommandozeilenparameter zum Starten des Defaultprogrammes für assoziierte Dateitypen an

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

- Definiert das Defaultprogramm ("rexx.exe") für assoziierte Dateitypen (assoziiert mit dem Bezeichner "rexxfile") und gibt die Argumente für das Programm an

Konfiguration von Dateien und ihren Anwendungen unter Windows, 3

- Laden/Ausführen von Dateien mit ihren Defaultprogrammen
  - Windows Explorer
    - Doppelklicken mit Maus auf Datei
    - Markieren der Datei und "Eingabe"- ("Enter"-) Taste drücken
    - Rechte Maustaste drücken und "Öffnen" auswählen
  - Kommandozeilenfenster
    - Geben Sie den Dateinamen *mit Dateiendung* an und drücken Sie die "Eingabe"- ("Enter"-) Taste

Konfiguration von Dateien und ihren Anwendungen unter Windows, 4

- Laden/Ausführen von Dateien mit ihren Defaultprogrammen
  - Kommandozeilenfenster
    - Dateiendung kann entfallen, *wenn* sie in der Umgebungsvariable "**PATHEXT**" angeführt ist
      - Aktuelle Wertebelegung im Kommandozeilenfenster darstellen

```
echo %pathext%
```

Ausgabe (vielleicht):

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

      - Neuen Wert der Umgebungsvariable "PATHEXT" hinzufügen

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

      - Für alle Sitzungen (Englisch: "Session")
        - Kontextmenü "Eigenschaften → Erweitert →Umgebungsvariablen" des Objektes "Arbeitsplatz" auswählen
        - Im unteren Fenster "Systemvariablen" den Eintrag "PATHEXT" auswählen, um die Zeichenkette ergänzen und speichern

Konfiguration von Dateien und ihren Anwendungen unter Windows, 5

- Abschließendes Beispiel
  - Definition einer eigenen Dateiendung für Object Rexx Programme

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

  - Nunmehr werden alle Dateien mit der Dateiendung ".recks" von Object Rexx (Defaultprogramm für "**recksDateien**") abgearbeitet, z.B. Eingabe in einem Kommandozeilenfenster:

```
meinTollesRexxProgramm.recks
```

  - Auf der Kommandozeile ist es nicht notwendig, für ".recks"-Dateien die Dateiendung anzugeben , z.B. Eingabe in einem Kommandozeilenfenster:

```
meinTollesRexxProgramm
```

Aufgabenstellungen und Präsentationen

- Gruppenarbeit bis zum nächsten Termin
  - Eine WSF-Datei mit drei enthaltenen (kurzen!) Jobs, die aus mindestens zwei Skripten bestehen
  - Eine WSC-Datei, die aus einem DHTML-Programm heraus benutzt wird
  - Automatisierung ("Fernsteuerung") von Windows Programmen
    - Mindestens drei Windows-Programme
    - Mindestens ein Windows-Programm darf nicht von Microsoft sein!
  - Präsentation der Beispiele
    - Maximal zwanzig Minuten Vortrag pro Gruppe, Diskussionszeit: fünf Minuten
    - Professionelle Folien (z.B. in OpenOffice)
    - *Jedes* Gruppenmitglied *muss* vortragen!