# Windows Script Host und ADITO

## AID 036 DE mit VB-Script

Autor: FA, MN, MW, KN Version 10.1. Zuletzt geändert 04.09.2017

| Version | Änderungen |
|---|---|
| **10.1** | Anpassung der Formatierungen |
| **10.0** | Anpassung an ADITO4.6 |
| **3.0** | Letzter Stand vor Übernahme in Versionierung |

# Inhaltsverzeichnis


**1. Einführung ..... 4**

1.1. Einsatzzweck der Schnittstelle ..... 4

1.2. Windows Script Host ..... 4

**2. Integration im Referenzsystem ..... 5**

2.1. Funktionsbibliotheken ..... 5

2.2. Ablauf ..... 5

2.3. Fehlerbehandlung ..... 5

**3. Einfaches Beispiel ..... 6**

**4. Word ..... 7**

4.1. Beispiel-Script ..... 7

**5. Excel ..... 8**

5.1. Parameter der Funktion ..... 8

5.1.1. dbAccess (Array, Pflichtfeld) ..... 8

5.1.2. headings (Array, Pflichtfeld) ..... 8

5.1.3. createPivot (Boolean, Pflichtfeld) ..... 9

5.1.4. startExcel (Boolean, Pflichtfeld) ..... 9

5.1.5. decChange (Boolean, optional) ..... 9

5.1.6. chartType (Integer, optional) ..... 9

**6. Outlook ..... 10**

6.1. Beispiel-Script ..... 10

# 1. Einführung

## 1.1. Einsatzzweck der Schnittstelle

Über eine Windows Script Host-Datei lassen sich per COM-Scripting beispielsweise Office-Applikationen steuern oder Systemeinstellungen verändern.

## 1.2. Windows Script Host

Auszug aus Wikipedia:

> Der **Windows Script Host** (**WSH**) (früher: **Windows Scripting Host**) ist in Windows-Betriebssystemen eine COM-basierte Laufzeitumgebung für Skriptsprachen.
>
> Der WSH lässt sich seit Windows 95 und Windows NT 4.0 verwenden, seit Windows 98 SE und Windows 2000 wird er standardmäßig installiert.
>
> Der Script Host wurde entwickelt, um Anwendern und insbesondere Administratoren die Möglichkeit zu geben, häufig wiederkehrende Vorgänge zu automatisieren. Während Unix, OS/2 und andere Systeme schon lange mit ausgereiften Programmiersprachen gesteuert werden konnten, boten die frühen Microsoft-Betriebssysteme (DOS mit und ohne Windows 3.x, Windows NT 3.5x) nur die sehr begrenzte Steuerungsmöglichkeit mittels Batch-Programmierung.
>
> Der WSH kann selbst keine Skripte ausführen, sondern verwendet sogenannte Script Engines. Diese Script Engines lassen sich wiederum in andere Programme bzw. Bibliotheken (z.B. Internet Explorer, Microsoft Office, Active Server Pages, ...) einbinden, wodurch diese ebenfalls zu Script Hosts werden. Mit dem WSH werden Script Engines für die Programmiersprachen JScript und VBScript mitgeliefert. Weitere Programmiersprachen für den WSH lassen sich nachinstallieren.

(Quelle: http://de.wikipedia.org/wiki/Windows_Script_Host)

Im ADITO-Referenzsystem werden an verschiedenen Stellen Windows-Scripts verwendet, um mit Microsoft Office-Anwendungen zu interagieren. Scriptsprache der Wahl ist hier VBScript (Visual Basic Script)

# 2. Integration im Referenzsystem

## 2.1. Funktionsbibliotheken

In den folgenden Funktionsbibliotheken werden Windows Scripts verwendet:

- **lib_excelex**
- **lib_wordbrf**
- **lib_email**

Zusätzlich wird die Schreibfähigkeit der Datei in der Funktionsbibliothek **lib_document** geprüft.

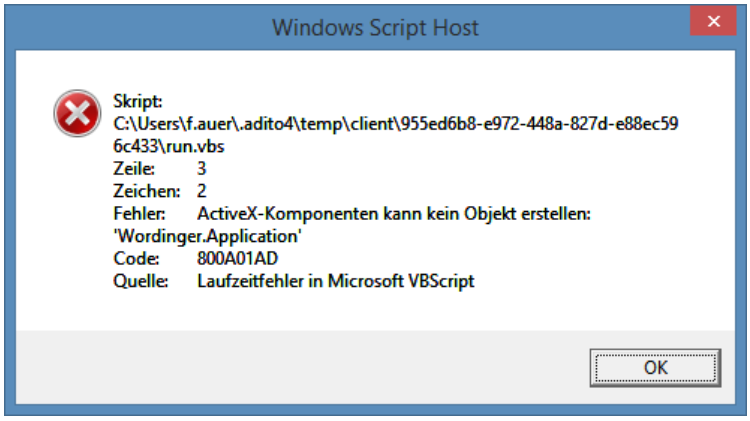
## 2.2. Ablauf

Im folgenden Ablauf wird mit dem Windows Script Host kommuniziert:

1. Erstellen des Scriptes als String in JDITO.
2. Anlegen der Datei (`run.vbs`) mit `swing.CLIENTCMD_STOREDATA`.
3. Ausführen der Datei mit `swing.CLIENTCMD_EXECUTE`.

## 2.3. Fehlerbehandlung

Tritt im Script selbst ein Fehler auf, erscheint die Fehlermeldung nicht über das ADITO-Fehlerlogging, sondern über die Windows-Fehlerkonsole:




In der Fehlermeldung wird angezeigt, in welcher Datei das temporäre Script ausgeführt wird. Hier kann die Analyse ("Zeile 3") durchgeführt werden. Das entsprechende Script wird in Kapitel 3 bearbeitet.

# 3. Einfaches Beispiel

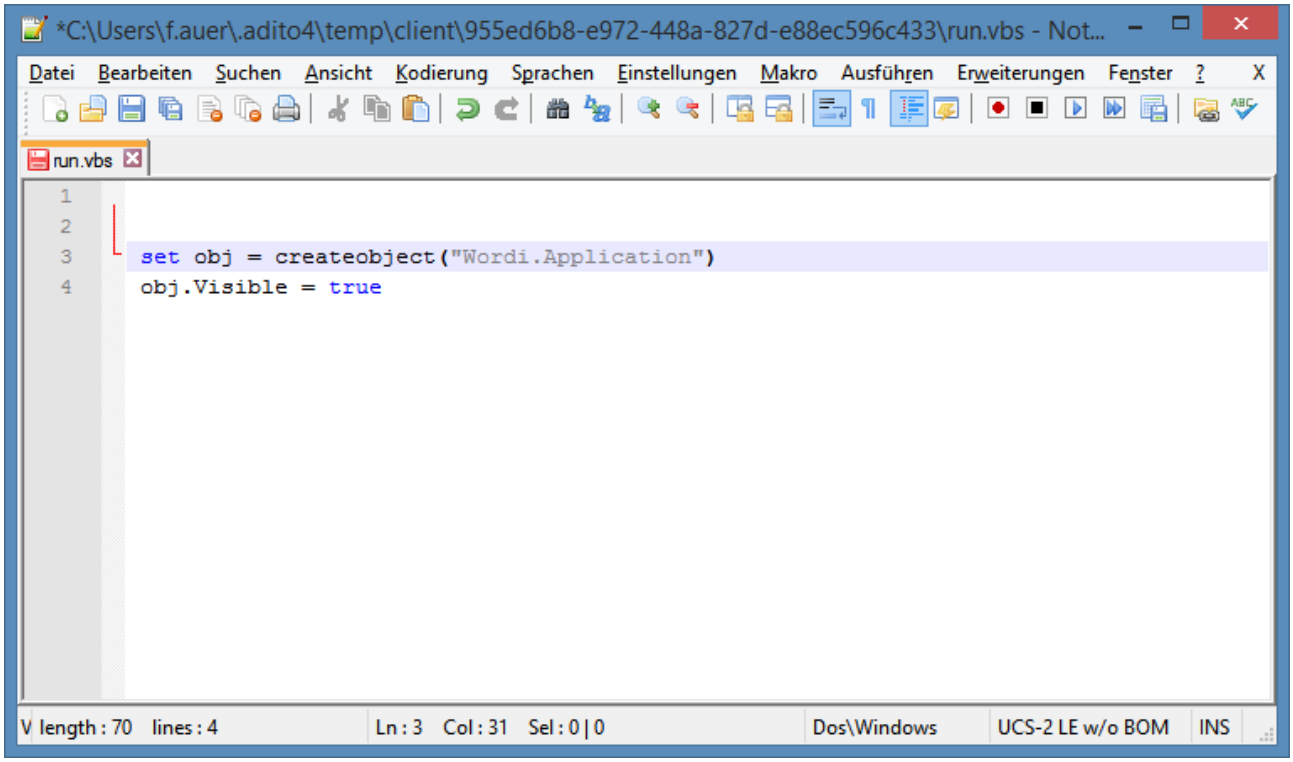
Das folgende Beispiel öffnet Word. Dazu wird nur die `lib_document` benötigt, welche die Hilfsfunktionen zum Speichern und wieder Öffnen der Datei anbietet.

```
import("lib_document");


var script = '\r\n '
+ '\r\n set obj = createobject("Word.Application")'
+ '\r\n obj.Visible = true';


var tempfile =  vars.getString("$sys.clienttemp") + "/" + "run.vbs";
FileIOwithError( swing.CLIENTCMD_STOREDATA, [ tempfile, script,
util.DATA_TEXT, false, "UTF-16LE"])
FileIOwithError( swing.CLIENTCMD_OPENFILE, [ tempfile ] )
```

Die Script-Datei, welche aus diesem Codestück generiert wird, sieht wie folgt aus:



# 4. Word

Die Word-Briefe werden als Beispiel-Datei in der Funktionsbibliothek `lib_wordbrf` in der Funktion `getScript` erstellt.

## 4.1. Beispiel-Script

```
    var script = '\r\n '
    + '\r\n set obj = createobject("Word.Application")'
    + '\r\n obj.Visible = true'
    + '\r\n dim fname'
    + '\r\n fname = "' + pFile + '"'
    + '\r\n obj.Documents.Open fname'
    + '\r\n set sel = obj.Selection'
    + '\r\n sel.Find.ClearFormatting'
    + '\r\n sel.Find.Replacement.ClearFormatting'
    + '\r\n dim txt\r\n';

    script += "\r\n";
    for (var i = 0; i < addrdata[0].length; i++)  // Kopfzeile
    {
        var str = addrdata[1][i];  // Datenzeile
        str = str.replace( new RegExp('"',"g" ), '""');
        str = str.replace( /\r\n/g , '" & vbCR & "');
        str = str.replace( /\n/g , '" & vbCR & "');
        str = str.replace( /\r/g , '" & vbCR & "');

        script += '\r\n sel.Find.Execute "@@' +  addrdata[0][i] +
'", false, false, false, false, false, true, 1, false, "'
        + str + '", 2, false, false, false, false'
    }
    return script;
```

In der For-Schleife werden die Exportfields aus Adressdaten (hier: Schlüsselwort "Exportfields") ersetzt.

# 5. Excel

Über den Scriptaufruf in der Funktionsbibliothek `lib_excelex` kann aus einer Datenbankabfrage eine Pivot-Tabelle mit Diagramm erstellt werden. Diese Datei wird dann als XHTML-Datei gespeichert, die nicht nur in Excel weiterbearbeitet werden kann, sondern auch zur Betrachtung in allen Browsern zur Verfügung steht.

## 5.1. Parameter der Funktion

`exportExcel()` benötigt beim Aufruf folgende Parameter, um ordnungsgemäß zu funktionieren:

```
function exportExcel(dbAccess, headings, createPivot, startExcel,
decChange, chartType)
```

### 5.1.1. dbAccess (Array, Pflichtfeld)

Ist ein zweidimensionales Array, das der Funktion übergeben wird. Prinzipiell kann die Funktion mit jedem beliebigen zweidimensionalen Array umgehen, in der Regel das Ergebnis einer SQL-Abfrage.

### 5.1.2. headings (Array, Pflichtfeld)

Ist ein eindimensionales Array, das der Funktion übergeben wird. Es besteht aus dem zukünftigen Dateinamen der Datei, den Titelzeilen und optional Steuerzeichen für die Microsoft Excel PivotTable. Da bei SQL-Abfragen nur die Datentabelle übergeben wird, nicht aber die Titel der Spalten, müssen die Spalten hier manuell benannt werden. Steuerzeichen werden den Spaltenüberschriften angehängt.

**Beispiel**: Das Ergebnis der SQL-Abfrage

```
SELECT PROJECTCODE, PROJECTSTATE, PROJECTTITLE, VOLUME FROM
PROJECTBODY
```

wird als Array „dbAccess" der Funktion `exportExcel()` übergeben. Um eine normale Tabelle (ohne PivotTable-Funktionen) zu erstellen, kann das Array `headings` wie folgt aussehen:

```
Projektbeispiel,Projektcode,Projektstatus,Projekttitel,Volumen
```

Es wird im von Adito voreingestellten Temp-Verzeichnis eine Datei „Projektbeispiel.html" erstellt. Die Spaltenüberschriften der Tabelle lauten „Projektcode", „Projektstatus", „Projekttitel", „Volumen".

Will man aus demselben Beispiel später eine PivotTable generieren (oder zumindest die Möglichkeit dazu haben), muss voreingestellt werden, wie die Spalten in der späteren PivotTable ausgerichtet werden sollen. Um die Ausrichtung zu bestimmen, wird an die Überschriften ein Suffix angehängt, das mit „@" beginnt. Das obige Beispiel kann dann so aussehen:

```
Projektbeispiel,Projektcode@2,Projektstatus@3,Projekttitel@1,Volumen
@4
```

Nach den „@“-Suffixen sind die Zahlen von 1 – 4 zulässig. Alle anderen Zahlen werden nicht beachtet.

- @1: Spalte wird am y-Feld der PivotTable angeordnet.
- @2: Spalte wird am x-Feld der PivotTable angeordnet.
- @3: Spalte ist ein Seitenfeld der PivotTable.
- @4: Spalte wird als Summenfeld der PivotTable betrachtet.

#### 5.1.3. createPivot (Boolean, Pflichtfeld)

Gibt an, ob aus dem Dokument eine PivotTable mit PivotChart-Bericht erstellt werden soll. Wird eine ungültige Ausrichtung im `headings`-Array übergeben, bricht Microsoft Excel die Auswertung mit einer Fehlermeldung ab. `createPivot` wird nur verwendet, wenn `startExcel` ebenfalls auf `true` gestellt ist.

#### 5.1.4. startExcel (Boolean, Pflichtfeld)

Gibt an, ob Microsoft Excel nach dem Erstellen der XHTML-Datei gestartet werden soll oder nicht.

#### 5.1.5. decChange (Boolean, optional)

Einige Datenbanken verwenden als Dezimaltrennzeichen einen Punkt (19.99) statt eines Kommas (19,99). Die deutsche Version von Microsoft Excel ist in der Regel jedoch so eingestellt, dass als Dezimaltrennzeichen nur ein Komma akzeptiert wird. Wird `decChange` mit `true` übergeben, so werden alle Punkte, die Dezimaltrennzeichen darstellen, zu Kommas umgewandelt.

#### 5.1.6. chartType (Integer, optional)

Gibt den Typen des im PivotChart-Bericht verwendeten Diagramms an. Eine Übersicht über die Diagrammtypen finden Sie bei der Beschreibung des Frames „Anlegen / Direkt Ausführen (Admin)“ hier im Dokument.

# 6. Outlook

In der Funktionsbibliothek `lib_email` wird ein Script erzeugt, welches Outlook öffnet und ein neues Mail-Objekt erstellt. Das ist notwendig, um eine formatierte HTML-Mail zu erstellen. Ohne diese Funktion kann nur eine Plain-Text-Mail über den URL-Aufruf mit "mailto:" erstellt werden.

## 6.1. Beispiel-Script

```
        var script =   'set MyApp =
CreateObject("Outlook.Application") \r\n'
        + 'set MyItem = MyApp.CreateItem(0) \r\n'
        + 'with MyItem \r\n';


        if (pRecipient != "" && pRecipient != undefined)
            script += '  .To = "' + getMailAddresses(pRecipient) + '"
\r\n';


        if (pCCRecipient != "" && pCCRecipient != undefined)
            script += '  .Cc = "' + getMailAddresses(pCCRecipient) +
'" \r\n';


        if (pBCCRecipient != "" && pBCCRecipient != undefined)
            script += '  .Bcc = "' + getMailAddresses(pBCCRecipient)
+ '" \r\n';


        if (pSubject != "" && pSubject != undefined)
            script += '  .Subject = "' + pSubject + '" \r\n';


        if (pText != "" && pText != undefined)
        {
            if ( pText.substr(0, 6) == "<html>" )   script += '
  .HTMLBody = "' +  mwspecialreplace(pText) + '" \r\n';
            else script += '  .Body = "' + mwspecialreplace(pText) +
'" \r\n';
        }


        if (pAttachments != undefined)
            for (var i = 0; i < pAttachments.length; i++)  script +=
'  .Attachments.Add "' + pAttachments[i] + '", olByValue, 1 \r\n';
```

```
        script += 'End With \r\n'
        + 'MyItem.Display';
```

In diesem Script wird zuerst Outlook mit einer neuen Mail erstellt, daraufhin werden Empfänger und Kopie-Empfänger befüllt. Schließlich erfolgt die Füllung mit HTML-Inhalt.