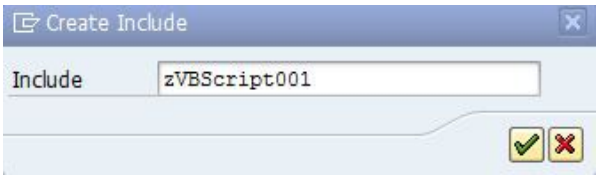
Der Beginn einer wunderbaren Freundschaft

von Stefan Schnell

Polyglottes programmieren, also programmieren in mehreren Sprachen, ist ein eher ungewöhnliches Szenario. Programmiersprachen bieten im Regelfall einen breiten Umfang, so dass ein Integrationsszenario mit einer weiteren Programmiersprache oftmals gar nicht notwendig ist. Jedoch kann hin und wieder der Wunsch oder sogar die Notwendigkeit entstehen, aus funktionalen oder schlichten ökonomischen Erwägungen, ein solches Integrationsszenario aufzubauen. Sei es weil die verwendete Programmiersprache kein adäquates Äquivalent liefert oder weil eine solche Implementierung zu aufwendig wäre.
In dieser beispielhaften Anleitung beschreibe ich die nahtlose Integration von VBScript® in ABAP®. Zum einen eröffnen wir uns damit die Möglichkeit auf einfachem Wege auf den Präsentationsserver zuzugreifen und zum anderen können wir die VBScript®-Sourcen direkt in die ABAP® Development Workbench integrieren.

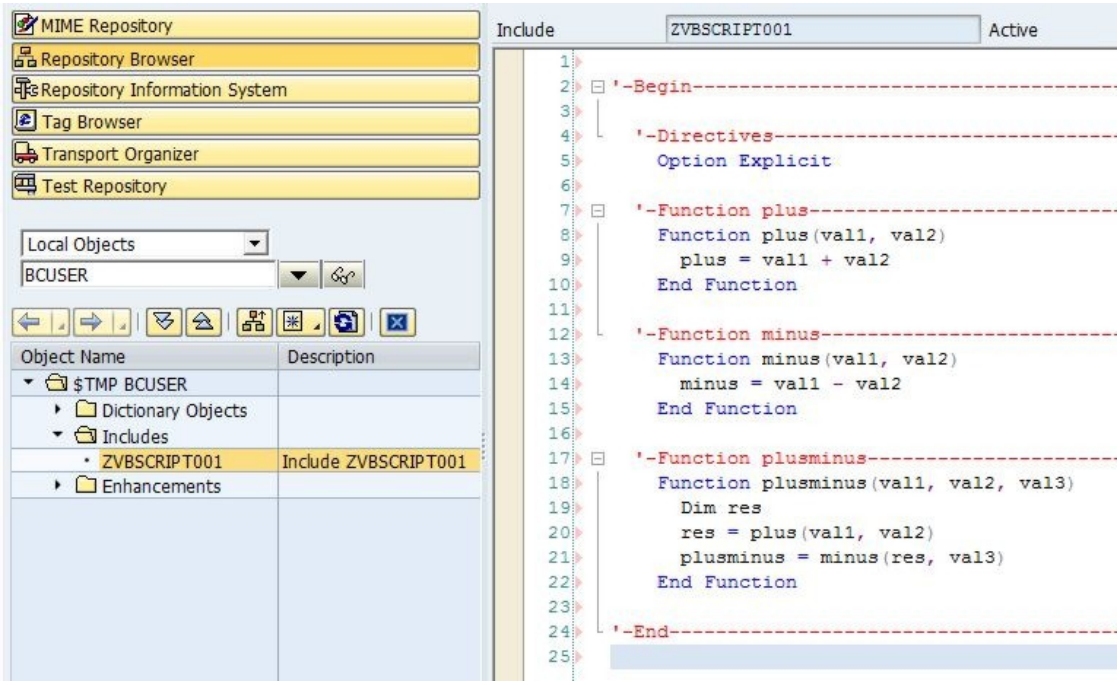
1. Wir beginnen damit, dass wir ein Include anlegen.



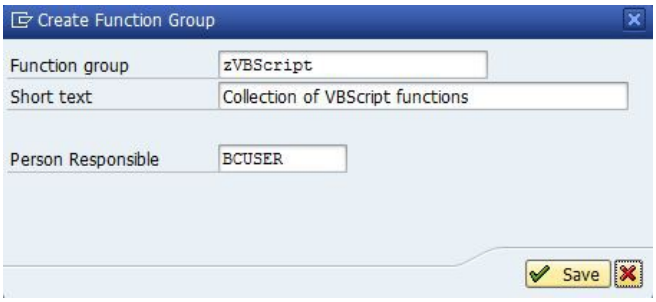
2. In dieses Include wird folgende VBScript®-Source implementiert:

```
'-Begin------------------------------------------------------------------

  '-Directives-----------------------------------------------------------
    Option Explicit

  '-Function plus--------------------------------------------------------
    Function plus(val1, val2)
      plus = val1 + val2
    End Function

  '-Function minus-------------------------------------------------------
    Function minus(val1, val2)
      minus = val1 - val2
    End Function

  '-Function plusminus---------------------------------------------------
    Function plusminus(val1, val2, val3)
      Dim res
      res = plus(val1, val2)
      plusminus = minus(res, val3)
    End Function

'-End--------------------------------------------------------------------
```

Es handelt sich hier um drei Funktionen, eine Additions-, eine Subtraktions- und eine Kombinationsfunktion aus beiden.
Am Ende das Aktivieren nicht vergessen. Ein Check führt selbstredend zu Fehlern. Sollte es beim Aktivieren zu Fehlern kommen, diese einfach ignorieren.

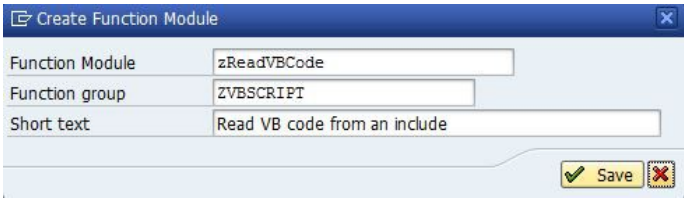


3. Im nächsten Schritt legen wir eine Funktionsgruppe an.

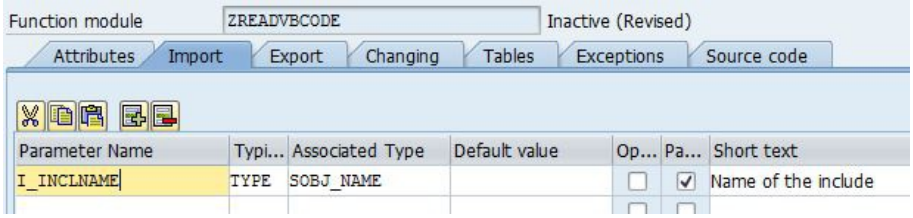


In dieser Funktionsgruppe sammeln wir alle Funktionen die unmittelbar mit der VBScript®-Verarbeitung in Verbindung stehen.

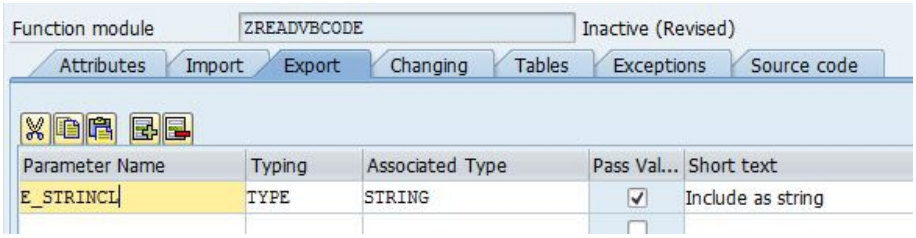
4. Nun legen wir in dieser neuen Funktionsgruppe einen Funktionsbaustein an.



5. Die Definition des Funktionsbausteins sieht wie folgt aus:



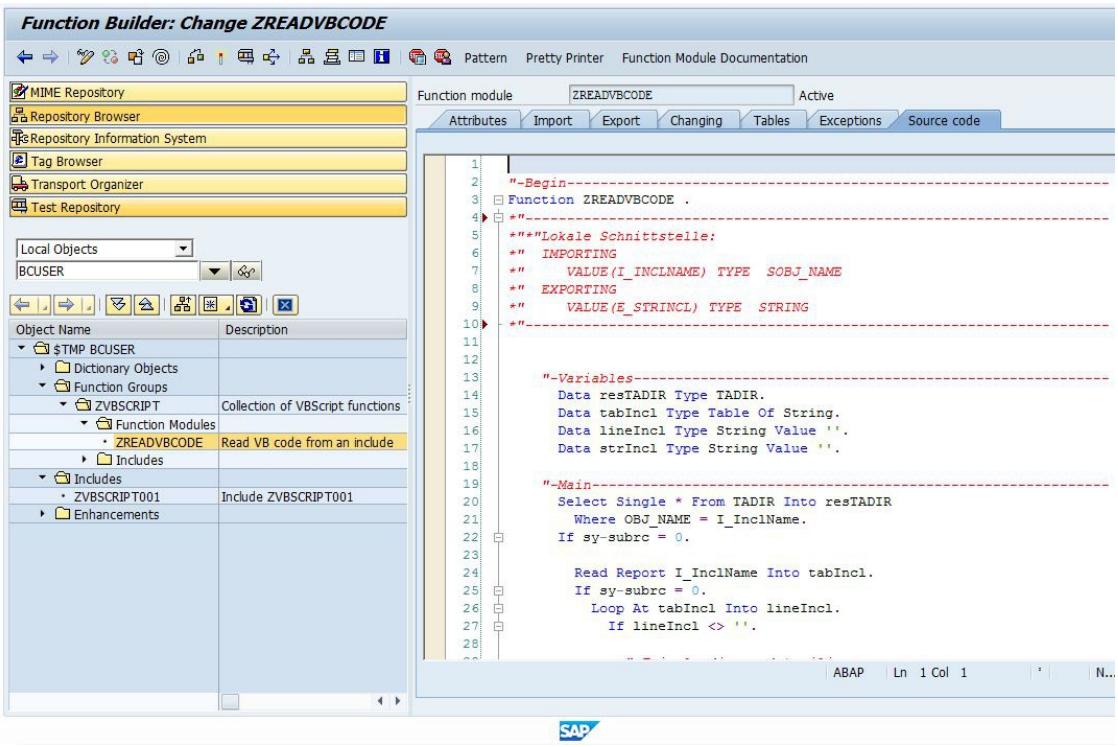
Der Beginn einer wunderbaren Freundschaft – VBScript® nahtlos mit ABAP® nutzen



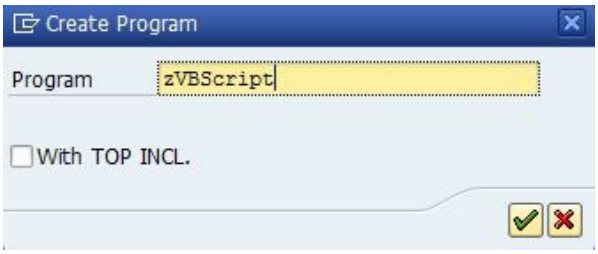
Es wird folgender ABAP®-Sourcecode implementiert:

```
"-Begin-----------------------------------------------------------------

  Function ZREADVBCODE .
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     VALUE(I_INCLNAME) TYPE  SOBJ_NAME
*"  EXPORTING
*"     VALUE(E_STRINCL) TYPE  STRING
*"----------------------------------------------------------------------

    "-Variables-------------------------------------------------------
      Data resTADIR Type TADIR.
      Data tabIncl Type Table Of String.
      Data lineIncl Type String Value ''.
      Data strIncl Type String Value ''.

    "-Main------------------------------------------------------------
      Select Single * From TADIR Into resTADIR
        Where OBJ_NAME = I_InclName.
      If sy-subrc = 0.

        Read Report I_InclName Into tabIncl.
        If sy-subrc = 0.
          Loop At tabIncl Into lineIncl.
            If lineIncl <> ''.

              "-Trim leading and trailing spaces-------------------------
                Condense lineIncl.

              "-If line is no comment------------------------------------
                If lineIncl+0(1) <> ''''.
                  Concatenate strIncl lineIncl
                    cl_abap_char_utilities=>cr_lf Into strIncl.
                EndIf.

              lineIncl = ''.

            EndIf.
          EndLoop.
        EndIf.

      EndIf.
      E_strIncl = strIncl.

  EndFunction.

"-End-------------------------------------------------------------------
```

Dieser Funktionsbaustein beinhaltet die Möglichkeit des Lesens von ABAP®-Programmen via Read Report. Die so eingelesenen Informationen werden als String-Ergebnis geliefert.



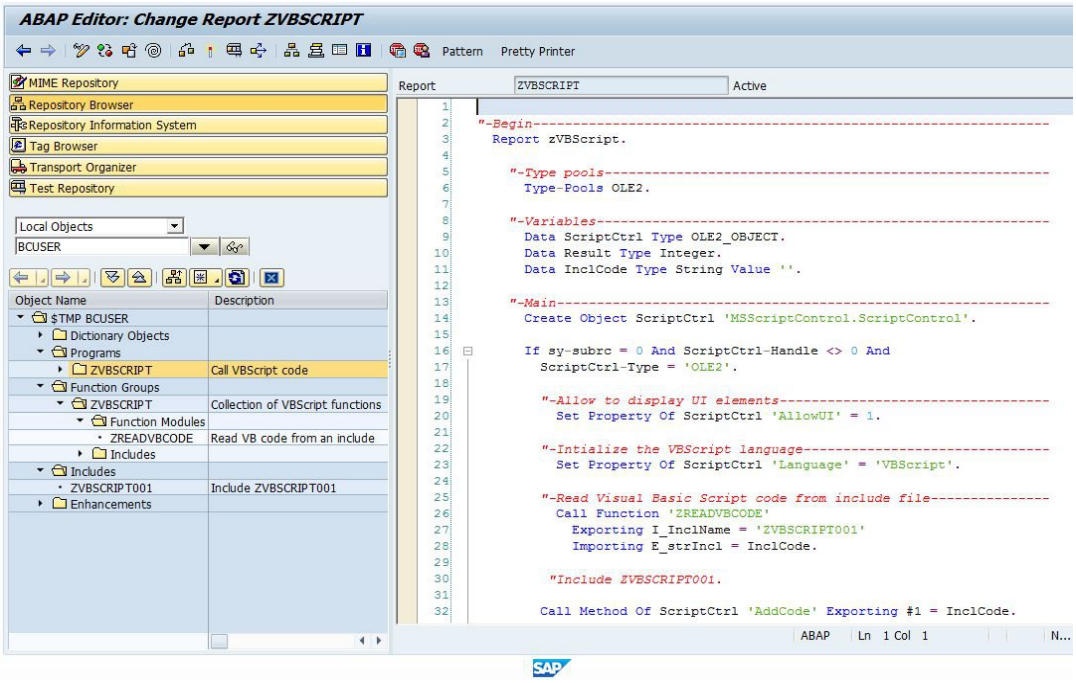
6. Last but not least legen wir ein ABAP®-Programm an.



7. Es wird folgender ABAP®-Sourcecode implementiert:

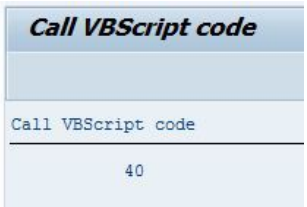
```
"-Begin----------------------------------------------------------------
  Program zVBScript.

    "-Type pools-----------------------------------------------------------
      Type-Pools OLE2.

    "-Variables------------------------------------------------------------
      Data ScriptCtrl Type OLE2_OBJECT.
      Data Result Type Integer.
      Data InclCode Type String Value ''.

    "-Main-----------------------------------------------------------------
      Create Object ScriptCtrl 'MSScriptControl.ScriptControl'.

      If sy-subrc = 0 And ScriptCtrl-Handle <> 0 And
        ScriptCtrl-Type = 'OLE2'.

        "-Allow to display UI elements-----------------------------------
          Set Property Of ScriptCtrl 'AllowUI' = 1.

        "-Intialize the VBScript language--------------------------------
          Set Property Of ScriptCtrl 'Language' = 'VBScript'.

        "-Read Visual Basic Script code from include file----------------
          Call Function 'ZREADVBCODE'
```

```
            Exporting I_InclName = 'ZVBSCRIPT001'
            Importing E_strIncl = InclCode.

          "Include ZVBSCRIPT001.

        Call Method Of ScriptCtrl 'AddCode' Exporting #1 = InclCode.

        If sy-subrc = 0.

          Call Method Of ScriptCtrl 'Eval' = Result
            Exporting #1 = 'plusminus(32, 16, 8)'.

          Write: / Result. "Result = 40

        EndIf.

        "-Free the object-------------------------------------------------
          Free Object ScriptCtrl.

      EndIf.

"-End--------------------------------------------------------------------
```



Das Programm instanziert die Klasse MSScriptControl und erhält damit Zugriff auf die VBScript®-Funktionalitäten. Es werden die Eigenschaften AllowUI und Language entsprechend gesetzt. Dann wird mittels des Funktionsbausteines zReadVBCode das Include mit dem VBScript®-Sourcecode eingelesen. Mit AddCode wird dieser dem Objekt bekannt gemacht und mit der Methode Eval wird die Funktion plusminus aufgerufen.

8. Das Resultat liefert das gewünschte Ergebnis.



Fazit

Wir wir sehen, ist eine nahtlose Integration der Scriptsprache VBScript® in ABAP® möglich. Sowohl die Implementierung des VBScript®-Sourcecodes als auch die Ausführung des VBScript®-Codes sind ohne Systembrüche möglich. Alles lässt sich über die ABAP® Development Workbench realisieren.
Und dies ist erst der Beginn einer wunderbaren Freundschaft. Mit diesem Ansatz lassen sich nun ohne Probleme weitere Sprachen einbinden. Z.B. kann über die Komponente ActiveXPoSH von SAPIEN® die vollständige PowerShell®-Funktionalität bereitgestellt werden und damit der ebenso vollständige Zugriff auf das Microsoft® dotNET®-Framework. So lassen sich auf selbigem Wege z.B. Visual Basic®-Funktionen implementieren – siehe die weiteren Aussichten.

Weiter Aussichten – Sonnig

1. Include mit VB#-Source:

```
'-Begin-----------------------------------------------------------------

  '-Directives------------------------------------------------------------
    Option Strict On

  '-Imports---------------------------------------------------------------
    Imports System
    Imports Microsoft.VisualBasic

  '-VBCode----------------------------------------------------------------
    Namespace nVBCode

      Public Class cVB

        Public Shared Function Hello1() As String
          Return "Hello World!"
        End Function

        Public Function Hello2(ByVal Name As String) As String
          Return "Hello " & Name & "!"
        End Function

        Public Sub Hello3(ByVal Name As String)
          MsgBox(Name, MsgBoxStyle.OkOnly, "Hello")
        End Sub

      End Class

    End Namespace

'-End-------------------------------------------------------------------
```

2. Das ABAP®-Programm dazu:

```
"-Begin-----------------------------------------------------------------
  Program zVBSharp.

    "-TypePools------------------------------------------------------------
      Type-Pools OLE2 .

    "-Constants------------------------------------------------------------
      Constants OUTPUT_CONSOLE Type i Value 0.
      Constants OUTPUT_WINDOW Type i Value 1.
      Constants OUTPUT_BUFFER Type i Value 2.

    "-Variables------------------------------------------------------------
      Data PS Type OLE2_OBJECT.
      Data Result Type i Value 0.
      Data strResult Type String Value ''.
      Data tabResult Type Table Of String.
      Data PSCode Type String Value ''.
      Data InclCode Type String Value ''.
```

```
"-Macros------------------------------------------------------------
  Define _.
    Concatenate PSCode &1 cl_abap_char_utilities=>cr_lf Into PSCode.
  End-Of-Definition.

"-Main--------------------------------------------------------------
  Create Object PS 'SAPIEN.ActiveXPoSH'.
  If sy-subrc = 0 And PS-Handle <> 0 And PS-Type = 'OLE2'.

    Call Method Of PS 'Init' = Result Exporting #1 = 0.
    If Result = 0.

      Call Method Of PS 'IsPowerShellInstalled' = Result.
      If Result <> 0.

        Set Property Of PS 'OutputMode' = OUTPUT_BUFFER.

        "-Read Visual Basic code from include file------------------
          Call Function 'ZREADVBCODE'
            Exporting I_InclName = 'ZVBSHARP001'
            Importing E_strIncl = InclCode.

        "-PowerShell Begin-----------------------------------------
          _ '$VBCode = @"'.
          "Include ZVBSHARP001.
          _ InclCode.
          _ '"@;'.

          _ 'Add-Type -TypeDefinition $VBCode -Language VisualBasic'.
          _ '$VB = new-Object nVBCode.cVB'.

          _ '[nVBCode.cVB]::Hello1()'.
          _ '$VB.Hello2("Stefan")'.
          _ '$VB.Hello3("Stefan")'.

          _ 'Remove-Variable VB'.
          _ 'Remove-Variable VBCode'.
          "-PowerShell End-------------------------------------------

        Call Method Of PS 'Execute' Exporting
          #1 = PSCode.

        Call Method Of PS 'OutputString' = strResult.

        Split strResult At cl_abap_char_utilities=>cr_lf
          Into Table tabResult.

        Loop At tabResult Into strResult.
          Write: / strResult.
        EndLoop.

      EndIf.

    EndIf.

    Free Object PS.
  EndIf.

"-End---------------------------------------------------------------
```

Urheberrechte

- SAP und ABAP sind eingetragene Warenzeichen und Eigentum der SAP AG
- Microsoft, VBScript, dotNET, PowerShell und Visual Basic sind eingetragene Warenzeichen und Eigentum von Microsoft
- SAPIEN ist Eigentum von SAPIEN