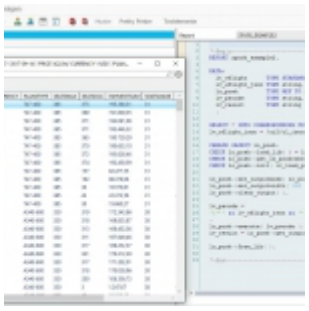
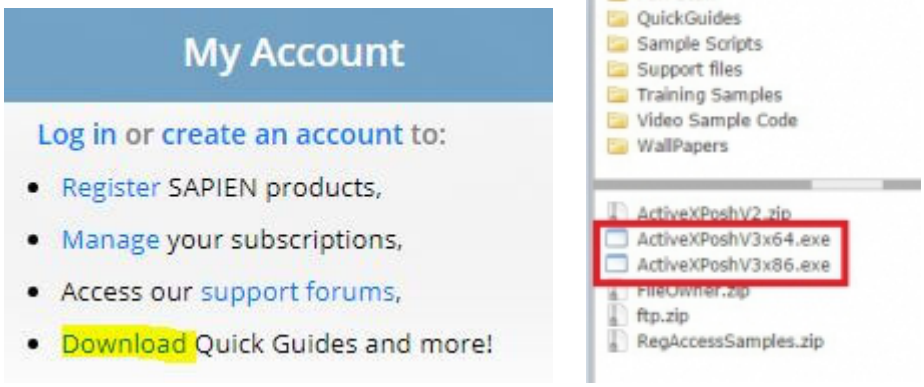


PowerShell mit ABAP nutzen

PowerShell ist eine sehr leistungsfähige Skriptsprache. Sie wird im Standard seit Windows 7 ausgeliefert. Aktuell ist Windows 10 mit PowerShell 5 ausgestattet. Der SAP GUI für Windows bietet eine COM-Schnittstelle die von ABAP aus verwendet werden kann. Mittels dieser Schnittstelle, und einer Bibliothek von SAPIEN, besteht die Möglichkeit das PowerShell auf dem Frontend-Server von ABAP genutzt werden kann. In diesem Beitrag soll kurz eine Realisierung mit einigen Anwendungsfällen umrissen werden.

Als erstes beschaffen wir uns die notwendigen Bibliotheken von SAPIEN. Einfach im Bereich My Account, nach dem Login, Download auswählen und die Bibliotheken herunter laden.



Nach der Installation und Registrierung können diese sofort mit ABAP verwendet werden. Für die Registrierung einfach eine Konsole im Administratormodus öffnen und die folgenden Befehle ausführen.

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\regasm.exe "C:\Program Files (x86)\SAPIEN Technologies, Inc\ActiveXPowerShell V3\\ActiveXPoshV3.dll" /codebase

C:\Windows\Microsoft.NET\Framework64\v4.0.30319\regasm.exe "C:\Program Files\SAPIEN Technologies, Inc\ActiveXPowerShell V3\ActiveXPoshV3.dll"
```

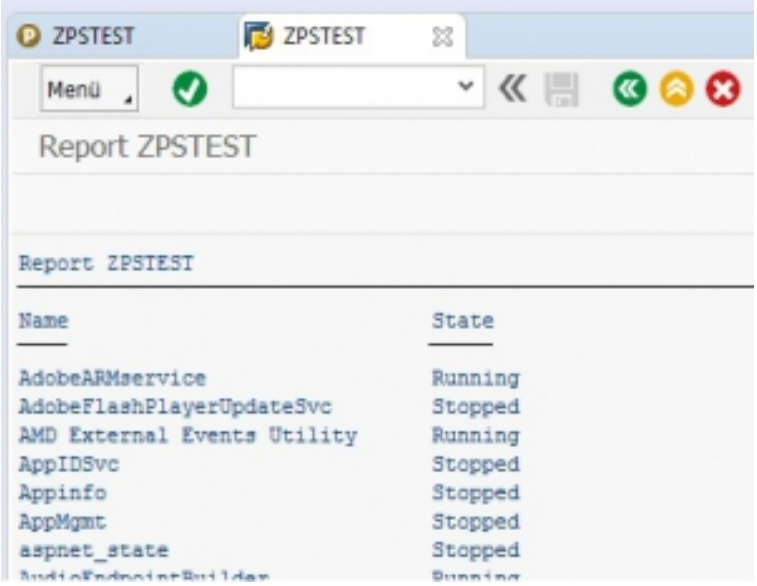
/codebase

Hier als Beispiel der Aufruf aller installierten Services mit ihrem Status.

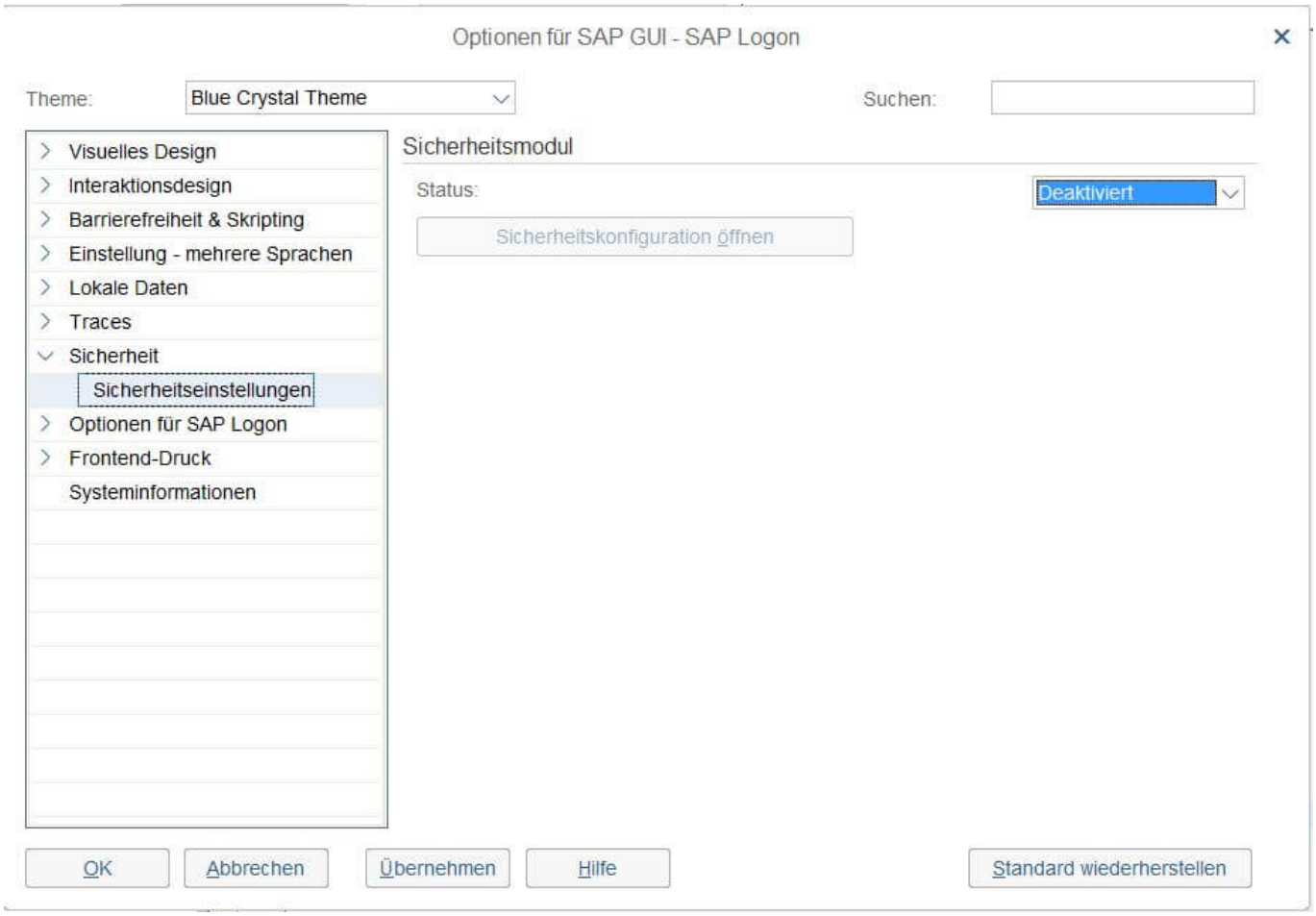
```
"-Begin-----------------------------------------------------------------
Program zPSTest.

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

  "-Variables-------------------------------------------------------------
  Data:
    PS Type OLE2_OBJECT,
    Result Type i,
    strResult Type String,
    tabResult Type Table Of String,
    cmd Type String
    .

  "-Main------------------------------------------------------------------
  Create Object PS 'SAPIEN.ActiveXPoSHV3'.
  Check sy-subrc = 0 And PS-Handle <> 0 Or PS-Type = 'OLE2'.

  Call Method Of PS 'Init' = Result Exporting #1 = 0.
  If Result <> 0.
    Free Object PS.
    Exit.
  EndIf.

  Call Method Of PS 'IsPowerShellInstalled' = Result.
  If Result = 0.
    Free Object PS.
    Exit.
  EndIf.

  Set Property Of PS 'OutputMode' = OUTPUT_BUFFER.

  cmd = `Get-WmiObject -class Win32_Service | `.
  cmd = cmd &&  `Format-Table -property Name,State`.

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

  Free Object PS.

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

In der Variablen cmd werden die PowerShell-Befehle übergeben, in diesem Fall die Ausgabe einer formatierten Tabelle mit Name und Status aller Services über das Windows Management Instrumentarium (WMI):
`Get-WmiObject -class Win32_Service | Format-Table -property Name,State`



Sollte der Aufruf nicht funktionieren, kann es sein, dass eine Standard-Sicherheitsregel den Aufruf blockiert. Zum Experimentieren können diese über den SAP Logon deaktiviert werden.



Um den kompletten Umfang der Bibliothek einfach nutzen zu können, hier eine entsprechende Klasse.

```
"-Begin----------------------------------------------------------------
class Z_CL_ACTIVEXPOSHV3 definition
  public
  create public .

  public section.

    constants MC_OUTPUTCONSOLE type I value 0 ##NO_TEXT.
    constants MC_OUTPUTWINDOW  type I value 1 ##NO_TEXT.
    constants MC_OUTPUTBUFFER  type I value 2 ##NO_TEXT.
    constants MC_TRUE          type I value 1 ##NO_TEXT.
    constants MC_FALSE         type I value 0 ##NO_TEXT.

  "! Loads the ActiveXPoshV3 library
  "!
  "! @parameter rv_result    | 1 for success, otherwise 0
  methods LOAD_LIB
    returning
      value(RV_RESULT) type I .

  "! Frees the ActiveXPoshV3 library
  methods FREE_LIB .
```

```
  "! Executes stored OLE activities
  methods FLUSH .

  "! Clears the internal output buffer
  "! when the OutputMode property is set to mc_OutputBuffer
  methods CLEAR_OUTPUT .

  "! Evaluates a PowerShell expression
  "! If the expression returns an object this function returns -1,
  "! otherwise 0. Output, if any, is not captured or redirected.
  "!
  "! @parameter iv_expression | PowerShell command
  "!
  "! @parameter rv_result     | Result of the command
  methods EVAL
    importing
      value(IV_EXPRESSION) type STRING
    returning
      value(RV_RESULT) type I .

  "! Executes a PowerShell command or script
  "! Output is directed according to the OutputMode property.
  "! Variable assignments persist between calls.
  "!
  "! @parameter iv_command    | PowerShell command or script
  methods EXECUTE
    importing
      value(IV_COMMAND) type STRING .

  "! Evaluates a PowerShell expression
  "!
  "! @parameter iv_expression | PowerShell command
  "!
  "! @parameter rv_result     | Value as string
  methods GET_VALUE
    importing
      value(IV_EXPRESSION) type STRING
    returning
      value(RV_RESULT) type STRING .

  "! Initial call to instantiate a PowerShell engine
  "! Required for any of the methods of this object to succeed.
  "!
  "! @parameter iv_load_profiles | Determines if your PowerShell profiles, if
they exist, are executed
  "!
  "! @parameter rv_result        | Returns 0 if successful, otherwise
&lt;&gt; 0
  methods INIT
    importing
      value(IV_LOAD_PROFILES) type I
```

```
    returning
      value(RV_RESULT) type I .

  "! Checks if PowerShell is installed
  "!
  "! @parameter rv_result       | Returns 1 if PowerShell is installed,
otherwise 0
  methods GET_IS_POWERSHELL_INSTALLED
    returning
      value(RV_RESULT) type I .

  "! Gets the current output mode
  "!
  "! @parameter rv_result       | 0 = OutputConsole, 1 = OutputWindow, 2 =
OutputBuffer
  methods GET_OUTPUTMODE
    returning
      value(RV_RESULT) type I .

  "! Sets the current output mode
  "!
  "! @parameter iv_mode         | 0 = OutputConsole, 1 = OutputWindow, 2 =
OutputBuffer
  methods SET_OUTPUTMODE
    importing
      value(IV_MODE) type I .

  "! Delivers the content of the output buffer as a single string
  "!
  "! @parameter rv_result       | Output buffer as string
  methods GET_OUTPUTSTRING
    returning
      value(RV_RESULT) type STRING .

  "! Gets the desired output width in characters
  "! PowerShell output often gets truncated, wrapped or adjusted
  "! corresponding to the width of a console window. Since there
  "! is not necessarily a console window available, the default
  "! is set to 80 characters width.
  "!
  "! @parameter rv_result       | Width in characters
  methods GET_OUTPUTWIDTH
    returning
      value(RV_RESULT) type I .

  "! Sets the desired output width in characters
  "! PowerShell output often gets truncated, wrapped or adjusted
  "! corresponding to the width of a console window. Since there
  "! is not necessarily a console window available, the default
  "! is set to 80 characters width.
  "!
```

```
  "! @parameter iv_width       | Width in characters
  methods SET_OUTPUTWIDTH
    importing
      value(IV_WIDTH) type I .

  "! Reads an include as string
  "!
  "! @parameter iv_incl_name   | Name of the include
  "!
  "! @parameter rv_str_incl    | Include as string
  methods READ_INCL_AS_STRING
    importing
      value(IV_INCL_NAME) type SOBJ_NAME
    returning
      value(RV_STR_INCL) type STRING .

  "! Loads a file from the MIME repository
  "! and copies it to the SAP GUI work directory
  "!
  "! @parameter iv_uri_file    | URI path of the file in the MIME repository
  methods LOAD_FILE_FROM_MIME
    importing
      !IV_URI_FILE type CSEQUENCE .

  "! Converts an outputstring to a string table
  "!
  "! @parameter iv_outputstring | String from get_outputstring
  "!
  "! @parameter rv_stringtable  | String in table
  methods OUTPUTSTRING_TO_TABLE
    importing
      value(IV_OUTPUTSTRING) type STRING
    returning
      value(RT_STRINGTABLE) type Z_TAB_STRING .

  PRIVATE SECTION.

    METHODS isactivex
      EXPORTING ev_result TYPE i.

    DATA olib TYPE ole2_object.

ENDCLASS.



CLASS Z_CL_ACTIVEXPOSHV3 IMPLEMENTATION.



  METHOD clear_output."-----------------------------------------------
```

```
  CALL METHOD OF olib 'ClearOutput'.
ENDMETHOD.



METHOD eval."-------------------------------------------------------
  CALL METHOD OF olib 'Eval' = rv_result
    EXPORTING #1 = iv_expression.
ENDMETHOD.



METHOD execute."----------------------------------------------------
  CALL METHOD OF olib 'Execute'
    EXPORTING
      #1 = iv_command.
ENDMETHOD.



METHOD flush."------------------------------------------------------
  CALL METHOD cl_gui_cfw=>flush.
ENDMETHOD.



METHOD free_lib."---------------------------------------------------
  FREE OBJECT olib.
ENDMETHOD.



METHOD get_is_powershell_installed."--------------------------------
  GET PROPERTY OF olib 'IsPowerShellInstalled' = rv_result.
ENDMETHOD.



METHOD get_outputmode."---------------------------------------------
  GET PROPERTY OF olib 'OutputMode' = rv_result.
ENDMETHOD.



METHOD get_outputstring."-------------------------------------------
  GET PROPERTY OF olib 'OutputString' = rv_result.
ENDMETHOD.



METHOD get_outputwidth."--------------------------------------------
```

```
  GET PROPERTY OF olib 'OutputWidth' = rv_result.
ENDMETHOD.



METHOD get_value."-----------------------------------------------------
  CALL METHOD OF olib 'GetValue' = rv_result
    EXPORTING #1 = iv_expression.
ENDMETHOD.



METHOD init."----------------------------------------------------------
  CALL METHOD OF olib 'Init' = rv_result
    EXPORTING #1 = iv_load_profiles.
ENDMETHOD.



METHOD isactivex."-----------------------------------------------------

  DATA hasactivex(32) TYPE c.

  ev_result = 0.
  CALL FUNCTION 'GUI_HAS_OBJECTS'
    EXPORTING
      object_model         = 'ACTX'
    IMPORTING
      return               = hasactivex
    EXCEPTIONS
      invalid_object_model = 1
      OTHERS               = 2.
  CHECK sy-subrc = 0 AND hasactivex = 'X'.
  ev_result = 1.

ENDMETHOD.



METHOD load_file_from_mime."-------------------------------------------

  DATA:
    lr_mr_api     TYPE REF TO if_mr_api,
    lv_filedata   TYPE xstring,
    lv_workdir    TYPE string,
    lv_filepath   TYPE string,
    lt_filename   TYPE STANDARD TABLE OF string,
    lv_filename   TYPE string,
    lt_dtab       TYPE TABLE OF x255,
    lv_len        TYPE i,
    lv_fileexists TYPE abap_bool
```

```
  .

SPLIT iv_uri_file AT '/' INTO TABLE lt_filename.
READ TABLE lt_filename INDEX lines( lt_filename ) INTO lv_filename.

CALL METHOD cl_gui_frontend_services=>get_sapgui_workdir
  CHANGING
    sapworkdir = lv_workdir
  EXCEPTIONS
    OTHERS     = 1.

lv_filepath = lv_workdir && '\' && lv_filename.

CALL METHOD cl_gui_frontend_services=>file_exist
  EXPORTING
    file   = lv_filepath
  RECEIVING
    result = lv_fileexists
  EXCEPTIONS
    OTHERS = 1.

CHECK lv_fileexists = abap_false.

IF lr_mr_api IS INITIAL.
  lr_mr_api = cl_mime_repository_api=>if_mr_api~get_api( ).
ENDIF.

CALL METHOD lr_mr_api->get
  EXPORTING
    i_url     = iv_uri_file
  IMPORTING
    e_content = lv_filedata
  EXCEPTIONS
    OTHERS    = 1.

CHECK sy-subrc = 0.

CALL FUNCTION 'SCMS_XSTRING_TO_BINARY'
  EXPORTING
    buffer        = lv_filedata
  IMPORTING
    output_length = lv_len
  TABLES
    binary_tab    = lt_dtab.

CALL FUNCTION 'GUI_DOWNLOAD'
  EXPORTING
    bin_filesize = lv_len
    filename     = lv_filepath
    filetype     = 'BIN'
  TABLES
```

```
      data_tab     = lt_dtab
    EXCEPTIONS
      OTHERS       = 1.

ENDMETHOD.



METHOD load_lib."-------------------------------------------------------

  DATA rc TYPE i VALUE 0.

  rv_result = 0.
  CALL METHOD me->isactivex IMPORTING ev_result = rc.
  CHECK rc = 1.
  CREATE OBJECT olib 'SAPIEN.ActiveXPoSHV3'.
  CHECK sy-subrc = 0 AND olib-handle <> 0 AND olib-type = 'OLE2'.
  rv_result = 1.

ENDMETHOD.



METHOD outputstring_to_table."-----------------------------------------

  FIELD-SYMBOLS:
    <lv_string> TYPE string
    .

  SPLIT iv_outputstring
    AT cl_abap_char_utilities=>cr_lf
    INTO TABLE rt_stringtable.
  "-Delete empty lines-------------------------------------------------
  LOOP AT rt_stringtable ASSIGNING <lv_string>.
    CHECK <lv_string> IS INITIAL.
    DELETE rt_stringtable.
  ENDLOOP.

ENDMETHOD.



METHOD read_incl_as_string."-------------------------------------------

  DATA:
    lt_tadir TYPE tadir,
    lt_incl TYPE TABLE OF string,
    lv_inclline TYPE string,
    lv_retincl TYPE string
    .
```

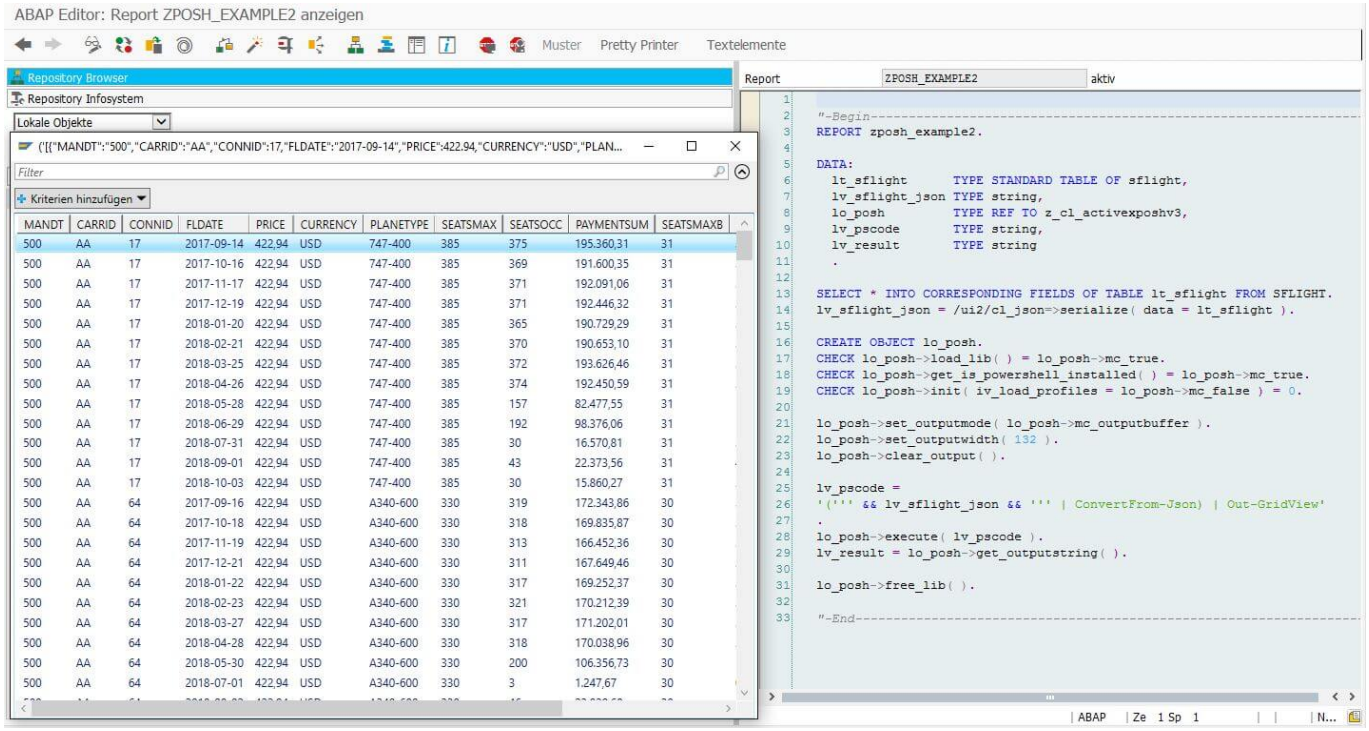
```
    SELECT SINGLE * FROM tadir INTO lt_tadir
      WHERE obj_name = iv_incl_name.
    CHECK sy-subrc = 0.
    READ REPORT iv_incl_name INTO lt_incl.
    CHECK sy-subrc = 0.
    LOOP AT lt_incl INTO lv_inclline.
      lv_retincl = lv_retincl && lv_inclline &&
        cl_abap_char_utilities=>cr_lf.
      CLEAR lv_inclline.
    ENDLOOP.
    rv_str_incl = lv_retincl.

  ENDMETHOD.



  METHOD set_outputmode."-----------------------------------------------
    SET PROPERTY OF olib 'OutputMode' = iv_mode.
  ENDMETHOD.



  METHOD set_outputwidth."----------------------------------------------
    SET PROPERTY OF olib 'OutputWidth' = iv_width.
  ENDMETHOD.



ENDCLASS.
"-End-----------------------------------------------------------------
```

Hier ein Beispielprogramm zum Anzeigen einer Tabelle im TableGrid von PowerShell. Die Daten werden einfach mittels eines Select gelesen und mit dem JSON-Serialisierer umgewandelt. Dann wird PowerShell geladen und initialisiert. Die JSON-Daten werden dem GridView nach einer Konvertierung über die Pipeline übergeben.

```
"-Begin---------------------------------------------------------------
REPORT zposh_example2.

DATA:
  lt_sflight      TYPE STANDARD TABLE OF sflight,
  lv_sflight_json TYPE string,
  lo_posh         TYPE REF TO z_cl_activexposhv3,
  lv_pscode       TYPE string,
  lv_result       TYPE string
  .

SELECT * INTO CORRESPONDING FIELDS OF TABLE lt_sflight FROM SFLIGHT.
lv_sflight_json = /ui2/cl_json=>serialize( data = lt_sflight ).

CREATE OBJECT lo_posh.
```

```
CHECK lo_posh->load_lib( ) = lo_posh->mc_true.
CHECK lo_posh->get_is_powershell_installed( ) = lo_posh->mc_true.
CHECK lo_posh->init( iv_load_profiles = lo_posh->mc_false ) = 0.

lo_posh->set_outputmode( lo_posh->mc_outputbuffer ).
lo_posh->set_outputwidth( 132 ).
lo_posh->clear_output( ).

lv_pscode =
'(''' && lv_sflight_json && ''' | ConvertFrom-Json) | Out-GridView'
.
lo_posh->execute( lv_pscode ).
lv_result = lo_posh->get_outputstring( ).

lo_posh->free_lib( ).

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

Und so sieht es dann aus.



Auf diese Art und Weise lassen sich nun sehr einfach Integrationsszenarien zwischen PowerShell und ABAP realisieren.

Hier ein Beispiel zum Anzeigen von .NET-Dialogen. Das Beispiel 1 öffnet einen Dialog, im Beispiel 2 wird das Ergebnis des 1. Dialoges angezeigt und im Beispiel 3 wird ein Datei-Öffnen-Dialog angezeigt und mit Auswahl eines Bildes wird dieses in einem eigenen Dialog angezeigt.

```
"-Begin----------------------------------------------------------------------
REPORT zposh_example1.

DATA:
  lo_posh         TYPE REF TO z_cl_activexposhv3,
```

```
  lv_pscode       TYPE string,
  lv_result       TYPE string
  .

CREATE OBJECT lo_posh.
CHECK lo_posh->load_lib( ) = lo_posh->mc_true.
CHECK lo_posh->get_is_powershell_installed( ) = lo_posh->mc_true.
CHECK lo_posh->init( iv_load_profiles = lo_posh->mc_false ) = 0.

lo_posh->set_outputmode( lo_posh->mc_outputbuffer ).
lo_posh->set_outputwidth( 132 ).
lo_posh->clear_output( ).

"-Example 1-----------------------------------------------------------
lv_pscode =
'[void][System.Reflection.Assembly]::LoadWithPartialName(`'              &&
'  "Microsoft.VisualBasic");'                                           &&
'[Microsoft.VisualBasic.Interaction]::MsgBox("Hello World", `'           &&
'  "YesNoCancel,SystemModal,Information", "Message");'.
lo_posh->execute( lv_pscode ).
lv_result = lo_posh->get_outputstring( ).
lo_posh->clear_output( ).

"-Example 2-----------------------------------------------------------
lv_pscode =
'[System.Windows.Forms.MessageBox]::Show("' && lv_result && '","Button",0);'.
lo_posh->execute( lv_pscode ).
lv_result = lo_posh->get_outputstring( ).
lo_posh->clear_output( ).

"-Example 3-----------------------------------------------------------
lv_pscode =
'Function GetFileName() {'                                               &&
'  $dlgOpen = New-Object System.Windows.Forms.OpenFileDialog;'           &&
'  $dlgOpen.Title = "Please select a file";'                             &&
'  $dlgOpen.InitialDirectory = `'                                        &&
'    "C:\Users\Public\Pictures\Sample Pictures\";'                       &&
'  $dlgOpen.Filter = "All Files (*.*)|*.*";'                             &&
'  $dlgOpen.ShowHelp = $True;'                                           &&
'  $result = $dlgOpen.ShowDialog();'                                     &&
'  If($result -eq "OK") {'                                               &&
'    [System.Diagnostics.Debug]::WriteLine($dlgOpen.FileName);'          &&
'    Return $dlgOpen.FileName;'                                          &&
'  }'                                                                    &&
'}'                                                                      &&

'[void][reflection.assembly]::LoadWithPartialName(`'                     &&
'  "System.Windows.Forms");'                                             &&
'$fileName = GetFileName;'                                               &&
'If ([string]::IsNullOrEmpty($fileName) -eq $True) { Exit }'             &&
'$file = (get-item $fileName);'                                          &&
```

```
'$img = [System.Drawing.Image]::Fromfile($file);'                    &&
'[System.Windows.Forms.Application]::EnableVisualStyles();'          &&
'$form = new-object Windows.Forms.Form;'                             &&
'$form.Text = "Image Viewer";'                                       &&
'$form.Width = $img.Size.Width;'                                     &&
'$form.Height =  $img.Size.Height;'                                  &&
'$pictureBox = new-object Windows.Forms.PictureBox;'                 &&
'$pictureBox.Width =  $img.Size.Width;'                              &&
'$pictureBox.Height =  $img.Size.Height;'                            &&
'$pictureBox.Image = $img;'                                          &&
'$form.controls.add($pictureBox);'                                   &&
'$form.Add_Shown( { $form.Activate() } );'                           &&
'$form.ShowDialog()'.
lo_posh->execute( lv_pscode ).
lv_result = lo_posh->get_outputstring( ).

lo_posh->free_lib( ).

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

In diesem Fall sind die PowerShell Sourcen direkt in den ABAP-Quellcode eingebettet. Diese können jedoch auch als Include-Entwicklungsobjekt gespeichert und mit der Methode READ_INCL_AS_STRING eingelesen und der Methode EXECUTE übergeben werden. Mit diesem Ansatz kann nun die gesamte .NET-Welt in ABAP integriert werden. Jede .NET-Bibliothek kann so genutzt werden und darüber hinaus können auch VB# resp. VB.NET und C# direkt integriert werden.