Powershell

Datenbanken und GUIs



Partnerarbeit von

Roland Jaggi, Joel Iselin, Michael Wettstein

Studiengang: Diplomausbildung HF Informatik

Fach: Scripting, 5.Semester

Schule: Technikerschule HF Zürich

Dozent: Lukas Müller

Beginn der Arbeit: 29.10.2020

Ende der Arbeit: 26.11.2020 Abgabe und Präsentation

# **Summary**

Ziel der Partnerarbeiten im Fach Skripting ist es, Einsicht in verschiedenen Powershell-Techniken und Möglichkeiten zu gewinnen, diese in einem praktischen Beispiel umzusetzen und zum Abschluss der Klasse zu präsentieren.

Das Thema unserer Gruppe ist «Datenbanken und grafische Benutzeroberflächen»

In diesem Bericht geben wir eine kurze Einführung und einen Überblick zu diesem Thema. In einem praktischen Beispiel mit Life Demo wird das Themengebiet der Klasse vorgestellt.

# Inhaltsverzeichnis

[Powershell 1](#_Toc56280977)

[Datenbanken und GUIs 1](#_Toc56280978)

[Summary 2](#_Toc56280979)

[Inhaltsverzeichnis 2](#_Toc56280980)

[1 Möglichkeiten und Einsatzbereiche 3](#_Toc56280981)

[2 ADO.NET Grundlagen 3](#_Toc56280982)

[3 Beispielprojekt 4](#_Toc56280983)

[4 Eingabemasken / WPF Powershell Kit (WPK) 5](#_Toc56280984)

[5 Schlusswort 6](#_Toc56280985)

[6 Quellenverzeichnis 7](#_Toc56280986)

[7 Abbildungsverzeichnis 7](#_Toc56280987)

[8 Tabellenverzeichnis 7](#_Toc56280988)

**Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.**

# Möglichkeiten und Einsatzbereiche

Powershell kann Beispielsweise von Datenbankadministratoren verwendet werden, um schnell und effizient Wartungsarbeiten und sich wiederholende Tätigkeiten zu automatisieren.

Datenbankzugriff per Powershell kann auch für einen Systemadministrator, der Beispielsweise in einem Büro für viele Rechner zuständig ist Vorteile bieten. Da Powershell auf jedem (Windows-) Rechner installiert ist, kann er seine Skripts auf jedem PC-Laufen lassen und bei Bedarf auch Pfade und Parameter individuell anpassen, ganz ohne Entwicklungsumgebung und ohne neu kompilieren zu müssen.

Powershell Skripts laufen auch auf Maschinen, auf denen keine SQL-Software installiert ist.

Unabhängig davon, ob Datenbankzugriff erforderlich ist oder nicht, ist ein grosser Vorteil von Powershell, dass die Skripts problemlos anderen Mitarbeitern und/oder Kunden zur Verfügung gestellt werden können. Beispielsweise für einmalige Konfigurationsanpassungen, um Installationen zu managen, um Anwendungen miteinander zu verknüpfen oder für temporäre Anwendungen, für die sich die Entwicklung einer Applikation nicht lohnt.

# ADO.NET Grundlagen

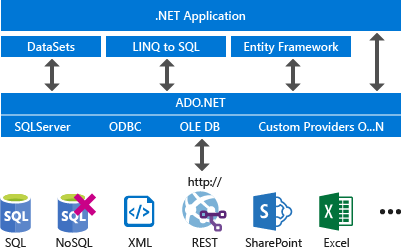
ADO.NET ist ein Teil der von Microsoft entwickelten .NET-Plattform. Es handelt sich um eine Sammlung von Klassen, die den Zugriff auf relationale Datenbanken gewährleisten. Von lokalen Microsoft Access Datenbanken und Excel-Spreadsheets über Microsoft SQL Server bis hin zu MySQL und Oracle Datenbanken.

Abbildung 1: Grundlagen ADO.NET

Da Powershell den Zugriff auf .NET-Klassen ermöglicht, kann ADO.NET auch über Powershell angesprochen werden.

# Beispielprojekt

## Kontextdiagramm

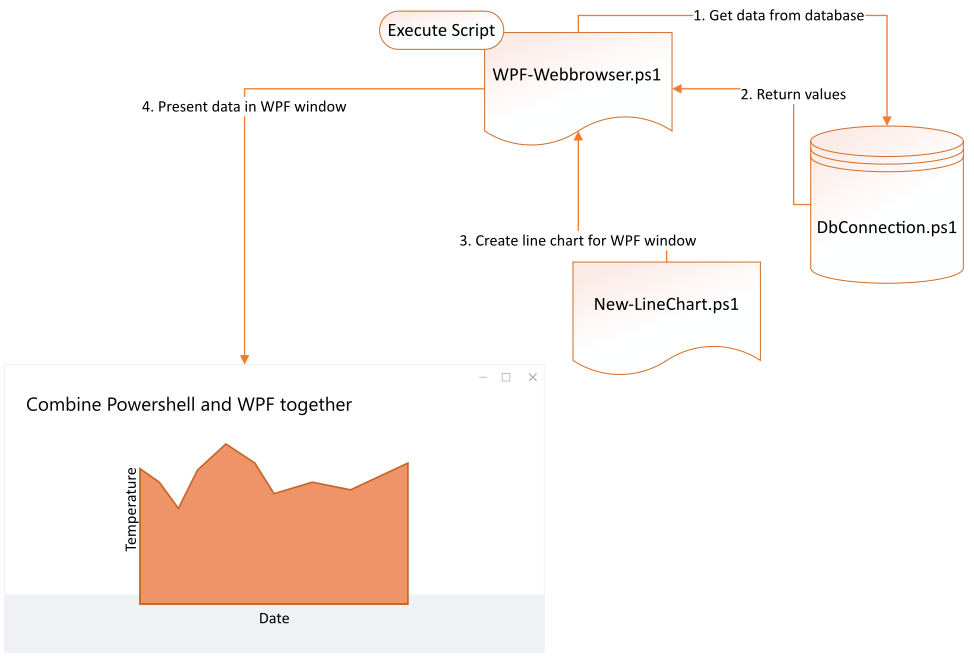


Abbildung 3: Kontextdiagramm

## Datenbankmodell

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 4: Datenbankmodell

## DbConnection.ps1

##########################################

# Created by Roland Jaggi, Joël Iselin, Michael Wettstein

# Description:

# Basic database connection to a MySQL database

##########################################

function GetTemperature {

$MySQLAdminUserName = 'air\_live'

$MySQLAdminPassword = 'juventus'

$MySQLDatabase = 'air\_live'

$MySQLHost = '127.0.0.1'

$ConnectionString = "server=" + $MySQLHost + ";port=3306;uid=" + $MySQLAdminUserName + ";pwd=" + $MySQLAdminPassword + ";database="+$MySQLDatabase

$Query = "SELECT temperature, MeasuredDateTime FROM(SELECT temperature, MeasuredDateTime FROM temperature ORDER BY MeasuredDateTime DESC LIMIT 10) AS TempAndDate ORDER BY MeasuredDateTime"

# Establish connection to the database and return values

Try {

# load connector

[void][System.Reflection.Assembly]::LoadWithPartialName("MySql.Data")

$Connection = New-Object MySql.Data.MySqlClient.MySqlConnection

# Establish connection

$Connection.ConnectionString = $ConnectionString

$Connection.Open()

# perform sql query

$Command = New-Object MySql.Data.MySqlClient.MySqlCommand($Query, $Connection)

$DataAdapter = New-Object MySql.Data.MySqlClient.MySqlDataAdapter($Command)

$DataSet = New-Object System.Data.DataSet

$RecordCount = $dataAdapter.Fill($dataSet, "data")

return $DataSet.Tables[0]

}

Catch {

Write-Host "ERROR : Unable to run query : $query `n$Error[0]"

}

Finally {

# Close open connection to the database

$Connection.Close()

}

}

# Eingabemasken / WPF Powershell Kit (WPK)

Eine sehr einfache Variante, um mit Powershell eine GUI zu erstellen bietet WPF-XAML. Das Layout kann dabei ähnlich einer HTML-Datei erstellt werden. Das kostenlose Visual Studio community bietet zudem die Möglichkeit, das Layout einfach in einem grafischen Editor zu erstellen.

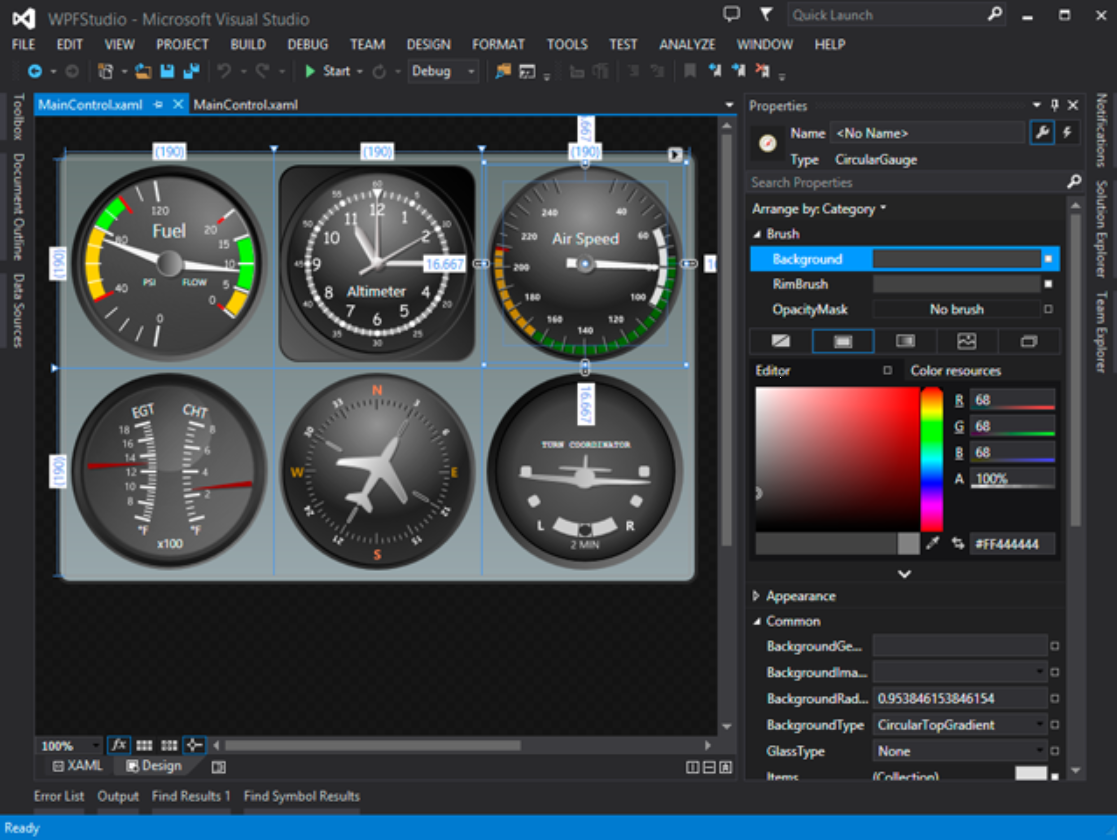


Abbildung 5 - Visual Studio Designer

Der XAML Code für das GUI wird von Visual Studio generiert und kann in das Powershell Skript kopiert werden. Der eigentliche Programmcode wird mit Powershell umgesetzt.

Es besteht auch die Möglichkeit XAML Code aus einem Powershell Skript in den Visual Studio Designer zu kopieren, um das Design anzupassen oder zu erweitern.

# WPF Code

## WPF-Webbrowser.ps1

##########################################

# Created by Roland Jaggi, Joël Iselin, Michael Wettstein

# Description:

# Starting point to display the WPF window

##########################################

#Load required libraries

Add-Type -AssemblyName PresentationFramework, PresentationCore, WindowsBase, System.Windows.Forms, System.Drawing

$ScriptPath = Split-Path -Parent $MyInvocation.MyCommand.Path

. .\DbConnection.ps1

# WPF Window to create basis layout

$Global:AllChartsPath = $ScriptPath

[xml]$xaml = @"

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfApp2"

Title="Combine Powershell and WPF together" Height="300" Width="300">

<Grid>

<Image Name='CornerImage' HorizontalAlignment="Left" VerticalAlignment="Top" Width="256" Margin="20,20,0,0" Opacity="0.5" />

<Label Content="Powershell WPF MySQL" HorizontalAlignment="Center" Margin="10,100,0,0" VerticalAlignment="Top" Width="500" HorizontalContentAlignment="Center" FontSize="24" Foreground="SlateGray"/>

<Label x:Name='TempLabel' Content="Latest 10 temperature inserts" HorizontalAlignment="Left" Margin="10,150,0,0" VerticalAlignment="Top" Width="396" HorizontalContentAlignment="Center" FontSize="18" Foreground="SlateGray"/>

<WebBrowser x:Name='WebBrowser3' HorizontalAlignment="Left" Height="500" Margin="25,180,0,0" VerticalAlignment="Top" Width="1050"/>

</Grid>

</Window>

"@

# Read the xaml grid and safe it to the reader

$Reader = (New-Object System.Xml.XmlNodeReader $xaml)

$Form = [Windows.Markup.XamlReader]::Load($Reader)

# Find all controlls in the xaml to be able to communicate with it

$xaml.SelectNodes("//\*[@\*[contains(translate(name(.),'n','N'),'Name')]]") | ForEach-Object {

New-Variable -Name $\_.Name -Value $Form.FindName($\_.Name) -Force

}

# Make juventus great again

$CornerImage.Source = "$ScriptPath\PsScripts\juventusLogo.png"

# Get path from the output folder

$HtmlCharts = "$ScriptPath\htmlCharts"

# Get temperature from the database

$GetTemperature = GetTemperature

$TemperatureValues = ""

$MeasuredDateTimeValues = ""

# loop trough the temperature and save them in a string to be able to display them on the chart

foreach($row in $GetTemperature)

{

$TemperatureValues += "$($row.Temperature),"

$tempDate = $row.MeasuredDateTime.ToString("yyyy-MM-dd HH:mm:ss") # Time formating for most Europeans

$MeasuredDateTimeValues += "'$($tempDate)',"

}

# Remove "," from the string otherwise it occures to an error because it will look for another value but does not find any

$TemperatureValues = $TemperatureValues.Substring(0,$TemperatureValues.Length-1)

$MeasuredDateTimeValues = $MeasuredDateTimeValues.Substring(0,$MeasuredDateTimeValues.Length-1)

# Pass values to the chart creation function

& "$ScriptPath\Psscripts\New-LineChart.ps1" -MeasuredDateTimeValues $MeasuredDateTimeValues -TemperatureValues $TemperatureValues -LegendLabel TemperatureLabel

$WebBrowser3.Navigate("file:///$HtmlCharts\Line.html")

# Window size

$Form.Height="850"

$Form.Width="1200"

# Mandetory last line of every script to load form

[void]$Form.ShowDialog()

## New-LineCart.ps1

##########################################

# Created by Roland Jaggi, Joël Iselin, Michael Wettstein

# Description:

# This script is creating line chart diagram

# by passing values

##########################################

# Define function parameters

Param

(

[parameter(Position=0, Mandatory=$true, ValueFromPipelineByPropertyName=$true)]

[string]$MeasuredDateTimeValues,

[parameter(Position=1, Mandatory=$true, ValueFromPipelineByPropertyName=$true)]

[string]$TemperatureValues,

[parameter(Position=2, Mandatory=$true, ValueFromPipeline=$true)]

[string]$LegendLabel

)

# Creating the html chart, add all the necessary js scripts to create the chart, using bootstrap for styling

$Chart = @"

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.6.0/Chart.min.js"></script>

<script src="JSScripts/Chart.min.js"></script>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

<link rel="stylesheet" href="JSScripts/bootstrap.min.css">

<title>Line Chart</title>

</head>

<body>

<div class="container">

<canvas id="lineChartDiagramm"></canvas>

</div>

<script>

let lineChartDiagramm = document.getElementById('lineChartDiagramm').getContext('2d');

Chart.defaults.global.defaultFontSize = 12;

Chart.defaults.global.defaultFontColor = '#777';

  let massPopChart = new Chart(lineChartDiagramm, {

type:'line',

data:{

labels:[$MeasuredDateTimeValues],

datasets:[{

label: `"$LegendLabel`" ,

data:[$TemperatureValues],

backgroundColor:'rgba(0,191,255, 0.6)',

hoverBackgroundColor: 'rgba(255,165,0, 0.6)',

borderWidth:1,

borderColor:'#777',

hoverBorderWidth:3,

hoverBorderColor:'#000',

fontsize:10,

}]

},

options:{

title:{

display:false,

fontSize:8

},

legend:{

display:false,

position:'bottom',

labels:{

fontColor:'Gray',

fontSize:12,

}

},

layout:{

padding:{

left:0,

right:0,

bottom:0,

top:0

}

},

tooltips:{

enabled:true,

titleFontSize:10

},

scales: {

yAxes: [{

ticks: {

beginAtZero: false,

display: true,

fontSize:10,

},

gridLines: {

display:true,

drawBorder: true,

}

}],

xAxes: [{

// Change here

barPercentage: 1,

ticks:{

fontSize: 12,

display: true,

},

gridLines: {

display:true,

drawBorder: true,

}

}]

}

}

});

</script>

</body>

</html>

"@

# Create html file with all the above informations

$Chart | Out-File -FilePath $Global:AllChartsPath\HtmlCharts\Line.html

# Schlusswort

Die Auseinandersetzung mit dem Thema war spannend und lehrreich. Das Erstellen des GUIs ging anfänglich erstaunlich leicht, bis dann alles genau so lief, wie es sollte waren dann aber doch viele Stunden nötig.

Skripten mit Powershell ist verglichen zum Programmieren mit einer vollwertigen Entwicklungsumgebung, wie wir es von Java oder C# her kennen, natürlich weniger komfortabel. Wenn etwas nicht funktioniert muss man sich selbst einen Debugger basteln, mit «Write-Output» oder temporären Infofeldern auf dem GUI.

Angenehm beim Entwickeln war hingegen, dass man keine Wartezeiten hatte für das Kompilieren und dass man das Skript Zeilenweise laufen lassen konnte. So konnte man sich manchmal in kleinen schnellen Schritten an das gewünschte Resultat herantasten.

Für grössere und komplexere Anwendungen wäre Powershell nicht das Werkzeug unserer Wahl, für kleinere Anwendungen und Tools ist es aber sicher eine Überlegung wert.

# Source Code

<https://github.com/loony/scriptingPowershellMySQLWPFChartDiagram>

# Quellenverzeichnis

|  |
| --- |
| https://www.admin-magazin.de/Das-Heft/2018/05/Datenbankzugriffe-mit-der-Powershell |
| https://www.script-example.com/Powershell-wpf-gui |
| https://dzone.com/articles/verify-database-access-with-Powershell-1 |
| https://www.script-example.com/powershell-wpf-gui |
| https://docs.microsoft.com/en-us/archive/msdn-magazine/2011/july/msdn-magazine-windows-powershell-with-wpf-secrets-to-building-a-wpf-application-in-windows-powershell |
| https://attilakrick.com/powershell/gui-mit-powershell-erstellen/ |

# **Abbildungsverzeichnis**

[Abbildung 1: Grundlagen ADO.NET 3](https://d.docs.live.net/83629ff0e1aae75d/2.%20Schule/2.%20Juventus/5.%20Semester/Projektarbeiten/4.%20Projekt%20Scripting/PA_Powershell_Datenbanken_Jaggi_Iselin_Wettstein.docx" \l "_Toc56337594)

[Abbildung 2 - .NET Übersicht / Quelle: www.microsoft.com 4](https://d.docs.live.net/83629ff0e1aae75d/2.%20Schule/2.%20Juventus/5.%20Semester/Projektarbeiten/4.%20Projekt%20Scripting/PA_Powershell_Datenbanken_Jaggi_Iselin_Wettstein.docx" \l "_Toc56337595)

[Abbildung 3: Kontextdiagramm 4](#_Toc56337596)

[Abbildung 4: Datenbankmodell 5](https://d.docs.live.net/83629ff0e1aae75d/2.%20Schule/2.%20Juventus/5.%20Semester/Projektarbeiten/4.%20Projekt%20Scripting/PA_Powershell_Datenbanken_Jaggi_Iselin_Wettstein.docx" \l "_Toc56337597)

[Abbildung 5 - Visual Studio Designer 7](#_Toc56337598)

c