

# IPI International

## *Working with PowerShell*

Name : IPI Modules and CRUD Operations  
Type : PowerShell Scripts and Modules  
Developer : Paul I Ighofose and Web Search Results  
Date Created : 16/01/2022  
Date Updated : 05/02/2022  
Requirements : Microsoft Windows 7 upwards. PowerShell version 5 up  
Function : PowerShell Query and CRUD Tool for csv files and SQL Server using either ACE OLEDB and System.Data.OleDb or System.Data.SqlClient connections. Also using JavaScript and JQuery to interact with PowerShell running WPF Forms and Web Browsers.

### **Introduction**

In my previous role as Senior Finance Analyst of NHS North West Surrey CCG, I used ADODB connections and VBA code written to pull data, build webpage displays to populate user forms from Outlook, Excel and Ms Access. That way I could get useful tabular data of an item selected in an Excel cell, MS Access field or Selected text of an Outlook item.

Occasionally, I just wanted a desktop link to give me quick access to structured result-sets. Having used PowerShell intensively for a lot of other quick access functions and knowing that it has many capabilities I 1<sup>st</sup> checked to see how to create VBA Types in PowerShell. Then I checked to see how to ADODB Record-sets. I found the PowerShell 3 videos with Jason Helmick and Jeffrey Snover encouraging and instructive about using PowerShell Modules.

I started with a simple Contacts record building and retrieval process to determine what was required to achieve CRUD operations, then display of the data from csv or SQL database sources, to finally populating Outlook items, PowerPoint Slides and word documents with filtered and non-filtered result-sets. With displaying the data, I was also able to achieve using both standard and Bootstrap CSS, JavaScript and JQuery, and get 2 way interaction with PowerShell.

### **Package Structure**

- A sample script file starts the process off, by offering test functions to create a Contacts Record and utilise all the Contacts Module functions. Both the sample script and the Contacts Module (TypeContact) utilise all other modules included
- There are 7 Modules
  - TypeContact
  - TypeWebForm
  - FormWPFWebView
  - CrudOledb
  - CrudSql
  - ScriptBlocks
  - SendTos

*Save the modules to the default PowerShell Modules folder and open the sample file in PowerShell ISE in Administrator mode*

---

### **Save Folders**

- Place the sample file **Test Contact Type Module.psm1** in \Documents\Source Files\ps1\  
This has functions to test each function of the TypeContact module
- Place the following files under \Documents\WindowsPowerShell\Modules\TypeContact\
  - TypeContact.psm1
  - CheckExists.sql
  - CreateTable.sql
  - spEntityCache\_Fix.sql
    - Place the files **psContacts.css** and **bootstrap.min.css** in \Documents\Source Files\css\
    - Place the files **psContacts.js** and **jquery-3.5.1.bom.js** in \Documents\Source Files\js\
- Place **TypeWebForm.psm1** in \ Documents\WindowsPowerShell\Modules\TypeWebForm\  
This module acts as the pre-cursor and build to the FormWPFWebView module. It bundles all the parameters passed to it in to a single object (much like a VBA Custom Type)
- Place **FormWPFWebView.psm1** in \Documents\WindowsPowerShell\Modules\FormWPFWebView\  
This module parses the XML from the module TypeWebForm and displays a WPF grid representing that input. It also displays a WebBrowser control
- Place **CrudOledb.psm1** in \Documents\WindowsPowerShell\Modules\CrudOledb\  
This module handles the select and filtering queries for csv files
- Place **CrudSql.psm1** in \Documents\WindowsPowerShell\Modules\CrudSql\  
This modules handles the full CRUD operations for interacting with SQL Server
- Place **ScriptBlocks.psm1** in \Documents\WindowsPowerShell\Modules\ScriptBlocks\  
This module contains adhoc functions for
  - Formating
  - Conversions
  - Parameter Retrieval
- Place **SendTos.psm1** in \Documents\WindowsPowerShell\Modules\SendTos  
This module handles all the Send to requests for send displayed results to
  - Outlook
  - PowerPoint
  - Word

---

## Sample File Structure

---

```
Test Contact Type Module.ps1* X
1 param ($First_Name, $Last_Name, $Phones, $Emails, $Contact_Title, $ShowSql, $ShowCsv)
2
3 $csv = '..\csv\psContacts.csv'
4 $html = '..\html\psContacts.htm'
5 $style = '<link media="screen" rel="stylesheet" href="..\css\psContacts.css" /><link rel="stylesheet" href="..\css\bootstrap.min.css" />'
6 $script = '<script src="..\js\psContacts.js"></script>'
7 $jQuery = '<script src="..\js/jquery-3.5.1.bom.js"></script>'
8 $title = 'Contacts'
9 $Server = '(LocalDB)\MSSQLLocalDB'
10 $Database = 'PowerShellModulesDb'
11 $Table = 'psContacts'
12
13 function Test-WebForm
14 { ... }
15
16 function Test-GetContactsCSV
17 { ... }
18
19 function Test-InsertCSV
20 { ... }
21
22 function Test-InsertOLEDBCSV
23 { ... }
24
25 function Test-UpdateCSV
26 { ... }
27
28 function Test-DeleteCSV
29 { ... }
30
31 function Test-ShowCSV
32 { ... }
33
34 function Test-GetContactsSQL
35 { ... }
36
37 function Test-InsertSQL
38 { ... }
39
40 function Test-UpdateSQL
41 { ... }
42
43 function Test-DeleteSQL
44 { ... }
45
46 function Test-ShowSQL
47 { ... }
48
49 if($ShowSql)
50 { ... }
51
52 if($ShowCsv)
53 { ... }
54
55 cd "C:\env\HOMEPATH\Documents\Source Files\ps1"
56 if (!$First_Name) {
57     . .\Test Contact Type Module.ps1 Paul Ighofose 01932,777 paul@home, paul@work 'Mr.'
58 }
59 }
```

- Running the sample file in Administrator Mode will load all its test functions for you to try out in the PowerShell console. It will also fill its parameters with the sample variables given for a new contact.
- The changing of the directory is **VERY** important as all the other modules are meant to use \Documents\Source Files\ directory as the base for file locations. As you can see beneath the last function (Test-SHOWSQL) the directory is changed to the \Documents\Source Files\ps1\
- So any csv file created by the modules will be stored in \Documents\Source Files\csv\ and html files in \Documents\Source Files\html\

```

Test Contact Type Module.ps1* X
1 param ($First_Name, $Last_Name, $Phones, $Emails, $Contact_Title, $ShowSql, $ShowCsv)
2
3 $csv = '..\csv\psContacts.csv'
4 $html = '..\html\psContacts.htm'
5 $style = '<link media="screen" rel="stylesheet" href="..\css\psContacts.css" /><link rel="stylesheet" href="..\css/bootstrap.min.css" />'
6 $script = '<script src="..\js\psContacts.js"></script>'
7 $jQuery = '<script src="..\js/jquery-3.5.1.bom.js"></script>'
8 $title = 'Contacts'
9 $Server = '(LocalDB)\MSSQLLocalDB'
10 $Database = 'PowerShellModulesDb'
11 $Table = 'psContacts'
12
13 function Test-WebForm
14 { ... }
15
16 function Test-GetContactsCSV
17 {
18     $tbl = (Convert-ContactCSVtoHTML -CsvPath $csv -HtmlPath $html -Head "$style`r$jQuery`r$script" -Title $title)
19     Invoke-item $html
20 }
21
22 function Test-InsertCSV
23 {
24     $test = Set-Contact -First_Name $First_Name -Last_Name $Last_Name -Phones $Phones -Emails $Emails -Contact_Title $Contact_Title
25     if ($test)
26     {
27         Insert-ContactCSV -CsvPath $csv -Object $test
28         Test-GetContactsCSV
29     }
30 }
31
32
33
34

```

- The Test-InsertCSV function will
  - Create a Record object using the TypeContact module and assign it to an object variable
  - Then pass in the parameters (updated when the file was first run) and the new object to Insert-ContactCSV of the TypeContact module
    - Which in turn converts the record and outputs its contents to a csv file in \Documents\Source Files\csv using PowerShell's Out-File cmdlet
  - Finally it calls the Test-GetContactCSV function above
    - Which in turn calls the Convert-ContactCSVtoHTML of the TypeContact module, passing in the csv location, html file output path and headers for the web page output
      - And the Convert-ContactCSVtoHTML retrieves the contents of the csv file and using the head and title parameters calls PowerShell's Convertto-HTML cmdlet to convert all to a html file to be saved in \Documents\Source Files\html\
    - And finally invokes the web page file created and displays it in your default web browser

```

25 function Test-InsertCSV
26 { ... }
34
35 function Test-InsertOLEDbCSV
36 {
37     $test = Set-Contact -First_Name $First_Name -Last_Name $Last_Name -Phones $Phones -Emails $Emails -Contact_Title $Contact_Title
38     if ($test)
39     {
40         Insert-ContactOLEDbToCSV -CsvPath $csv -Object $test
41         Test-GetContactsCSV
42     }
43 }

```

- The Test-InsertOLEDbCSV function does the same as Test-InsertCSV, except
  - The Insert-ContactOLEDbToCSV of the TypeContact module then uses the Insert-OLEDbCSV function of the CrudOledb module to insert the record using PowerShell's OleDb library

```

44
45 function Test-UpdateCSV
46 {
47     Param($Id = 1)
48     $test = Set-Contact -First_Name $First_Name -Last_Name $Last_Name -Phones $Phones -Emails $Emails -Contact_Title $Contact_Title
49     if ($test)
50     {
51         Update-ContactCSV -Id $Id -CsvPath $csv -Object $test
52         Test-GetContactsCSV
53     }
54 }
55
56 function Test-DeleteCSV
57 {
58     Param($Id = 2)
59     Delete-ContactCSV -Id $Id -CsvPath $csv
60     Test-GetContactsCSV
61 }

```

- Instead of using variables like \$First\_Name and so on you can just use string entries
- The -Phones and -Emails parameters take arrays so a comma between each entry will pass them in as a string
- Or you can run the whole file with the string entries (the first . (dot and a space) tells PowerShell to use the current session) notice (. .\)
- ..\Test Contact Type Module.ps1' Paul Ighofose 01932,777 paul@home, paul@work 'Mr.'
- As -First\_Name, -Last\_Name and -Contact\_Title all expect single string entries, you will have to surround any string with spaces that you want to just appear in one of these fields.
- Also notice that the entries are positional and without need for parameter names as long as they follow the right order.
- The Test-UpdateSQL function has a default parameter set to 1. Therefore to update a different record you would, in the console
  - Change any one of \$First\_Name, \$Last\_Name, \$Phones, \$Emails, \$Contact\_Title variables
  - Enter Test-UpdateSQL 2 (the Id number of the current record you are amending)
- The Id parameter is only used in Delete and Update functions to specify which record to update as the insert functions only use a generated Id

```

67
68 function Test-GetContactsSQL
69 {
70     $tbl = (Convert-ContactSQLtoHTML -Server $Server -Database $Database -Table $Table -HtmlPath $html -Head "$style`r`njQuery`r`n$script" -Title $title)
71     if ($tbl)
72     {
73         Invoke-item $html
74     }
75 }
76
77 function Test-InsertSQL
78 {
79     $test = Set-Contact -First_Name $First_Name -Last_Name $Last_Name -Phones $Phones -Emails $Emails -Contact_Title $Contact_Title
80     if ($test)
81     {
82         Insert-ContactSQL -Server $Server -Database $Database -Table $Table -Object $test
83         Test-ShowSQL
84     }
85 }
86
87 function Test-UpdateSQL
88 {
89     Param($Id = 1)
90     $test = Set-Contact -First_Name $First_Name -Last_Name $Last_Name -Phones $Phones -Emails $Emails -Contact_Title $Contact_Title
91     if ($test)
92     {
93         Update-ContactSQL -Id $Id -Server $Server -Database $Database -Table $Table -Object $test
94         Test-ShowSQL
95     }
96 }
97
98 function Test-DeleteSQL
99 {
100     Param($Id = 2)
101     Delete-ContactSQL -Id $Id -Server $Server -Database $Database -Table $Table
102     Test-ShowSQL
103 }
104
105 function Test-ShowSQL
106 {
107     Show-ContactSQL -Server $Server -Database $Database -Table $Table -Head "$style`r`njQuery`r`n$script" -Title $title
108 }
--

```

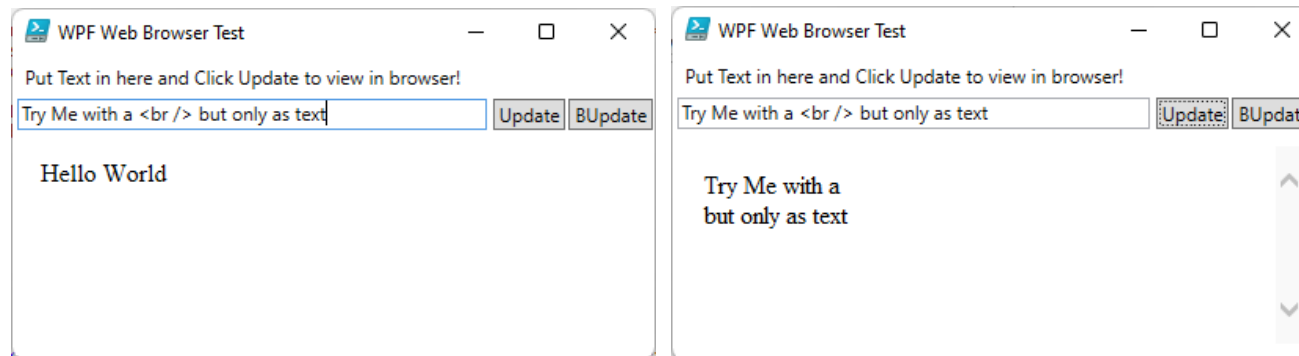
- The Test-InsertSQL, Test-UpdateSQL, Test-DeleteSQL all work similarly to the CSV functions with the exception that
  - When Test-InsertSQL is run
    - The TypeContact module
      - Checks if a Database for the name given exists and if it has a table of the name given. If not, it requests confirmation from the user to create a database and table with the names given.
      - This is where the additional .sql files in the TypeContact folder play their part. The names in those scripts for the database and table are replaced by the names you specify in your parameters. The stored procedure fixes the database scoped configuration of IDENTITY\_CACHE to ensure that when records are added, there are no jumps in Id number like the last record having Id = 5 and the next record having Id = 1001.
      - The default database name is PowerShellModulesDb
      - The default table name is psContacts
  - All CRUD operations are done using the CrudSql module
  - Instead of using Test-GetContactsSQL to retrieve the records from SQL Server and display them in your default web browser, they use Test-ShowSQLto to make a call to the Show-ContactSQL function of the TypeContact module

```

12
13 function Test-WebForm
14 {
15     $object = Set-WebForm
16     Show-WPFWebViewForm -object $object
17 }

```

- The Test-WebForm just loads the initial Proof of Concept test I ran when starting.
- It calls the Set-WebForm function of the TypeWebForm module
  - Which builds an object with parameters containing a couple of textboxes and buttons to test function and interaction between them and the WebBrowser control.
  - Then calls the Show-WPFWebViewForm function of the FormWPFWebView to parse the XML and display an interactive form.



```

62
63 function Test-ShowCSV
64 {
65     Show-ContactCSV -CsvPath $csv -HtmlPath $html -Head "$style`r$jQuery`r$script" -Title $title
66 }
67
68 function Test-GetContactsSQL
69 {
70     ...
71 }
72
73 function Test-InsertSQL
74 {
75     ...
76 }
77
78 function Test-UpdateSQL
79 {
80     ...
81 }
82
83 function Test-DeleteSQL
84 {
85     ...
86 }
87
88 function Test-ShowSQL
89 {
90     Show-ContactSQL -Server $Server -Database $Database -Table $Table -Head "$style`r$jQuery`r$script" -Title $title
91 }
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109

```

- The Show-ContactCSV and Show-ContactSQL functions in the TypeContact Module both pass parameter variables to and call the Set-WebForm function of the TypeWebForm module

```

1109
1110 function Show-ContactCSV
1111 {
1112     [CmdletBinding()]
1113     Param (
1114         [Parameter(Mandatory=$true,
1115             ValueFromPipeline=$true,
1116             ValueFromPipelineByPropertyName=$true,
1117             Position=0)]
1118         [ValidateNotNullOrEmpty()]
1119         [string]
1120         $CsvPath,
1121         [Parameter(Mandatory=$true,
1122             ValueFromPipeline=$true,
1123             ValueFromPipelineByPropertyName=$true,
1124             Position=1)]
1125         [ValidateNotNullOrEmpty()]
1126         [string]
1127         $HtmlPath,
1128         [Parameter(Mandatory=$false,
1129             ValueFromPipeline=$true,
1130             ValueFromPipelineByPropertyName=$true,
1131             Position=2)]
1132         [string]
1133         $Head,
1134         [Parameter(Mandatory=$false,
1135             ValueFromPipeline=$true,
1136             ValueFromPipelineByPropertyName=$true,
1137             Position=3)]
1138         [string]
1139         $Title,
1140         [Parameter(Mandatory=$false,
1141             ValueFromPipeline=$true,
1142             ValueFromPipelineByPropertyName=$true,
1143             Position=4)]
1144         [bool]
1145         $HtmlOnly = $false,
1146         [Parameter(Mandatory=$false,
1147             ValueFromPipeline=$true,
1148             ValueFromPipelineByPropertyName=$true,
1149             Position=5)]
1150         [psobject]
1151         $Filter
1152     )
1153     Begin
1154     { ... }
1155     Process
1156     { ... }
1157     End
1158     { ... }
1159 }

```

```

1286 function Show-ContactSQL
1287 {
1288     [CmdletBinding()]
1289     Param (
1290         [Parameter(Mandatory=$true,
1291             ValueFromPipeline=$true,
1292             ValueFromPipelineByPropertyName=$true,
1293             Position=0)]
1294         [ValidateNotNullOrEmpty()]
1295         [string]
1296         $Server,
1297         [Parameter(Mandatory=$true,
1298             ValueFromPipeline=$true,
1299             ValueFromPipelineByPropertyName=$true,
1300             Position=1)]
1301         [ValidateNotNullOrEmpty()]
1302         [string]
1303         $Database,
1304         [Parameter(Mandatory=$true,
1305             ValueFromPipeline=$true,
1306             ValueFromPipelineByPropertyName=$true,
1307             Position=2)]
1308         [ValidateNotNullOrEmpty()]
1309         [string]
1310         $Table,
1311         [Parameter(Mandatory=$false,
1312             ValueFromPipeline=$true,
1313             ValueFromPipelineByPropertyName=$true,
1314             Position=3)]
1315         [string]
1316         $Head,
1317         [Parameter(Mandatory=$false,
1318             ValueFromPipeline=$true,
1319             ValueFromPipelineByPropertyName=$true,
1320             Position=4)]
1321         [string]
1322         $Title,
1323         [Parameter(Mandatory=$false,
1324             ValueFromPipeline=$true,
1325             ValueFromPipelineByPropertyName=$true,
1326             Position=5)]
1327         [bool]
1328         $HtmlOnly = $false,
1329         [Parameter(Mandatory=$false,
1330             ValueFromPipeline=$true,
1331             ValueFromPipelineByPropertyName=$true,
1332             Position=6)]
1333         [psobject]
1334         $Filter
1335     )
1336     Begin
1337     { ... }
1338     Process
1339     { ... }
1340     End
1341     { ... }
1342 }

```

- The Show-ContactCSV and Show-ContactSQL functions in the TypeContact Module are almost identical with the following exceptions
  - That they call functions that are either csv or SQL related
  - The CSV element can load from an html file as well as object containing html, whereas the SQL element always has to load from an object containing html



```

1152 )
1153 Begin
1154 {
1155     if (!$Filter)
1156     {
1157         $obj = Get-ContactCSV -CsvPath $CsvPath
1158     }
1159     else
1160     {
1161         $obj = Get-ContactOLECSV $CsvPath -Filter $Filter
1162     }
1163 }
1164 Process
1165 {...}
1268 End
1269 {...}
1284 }

```

- The CSV element
  - loads the csv file using PowerShell's Import-Csv cmdlet when there is no filter
  - or when there is a filter, calls the Get-ContactOLECSV function
    - which builds the required SQL syntax for that filter
    - then calls the Get-OLEDbCSV of the CrudOledb module to run the command
      - which returns a Table from System.Data.DataSet

```

1335 )
1336 Begin
1337 {
1338     $obj = Get-ContactSQL -Server $Server -Database $Database -Table $Table -Filter $Filter
1339 }
1340 Process
1341 {...}
1444 End
1445 {...}
1460 }

```

- The SQL element
  - Calls the Get-ContactSQL function
    - Which builds the required SQL syntax with or without the filter
    - Then calls the CRUD-SQL function of the CrudSql module to run the command
      - Which returns a System.Data.DataTable

```

Process
{
    $body = (Format-Html $obj)
    $arr = @('..css/', '..js/')
    foreach ($itm in $arr)
    {
        $Head = ($Head -replace $itm, (Resolve-Path $itm))
    }
    $rDefs = '
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
    '
    $cDefs = '
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="Auto" />
    '
    $fields = @(
        @('Id', 60),
        @('Contact_Title', 80),
        @('First_Name', 120),
        @('Last_Name', 200),
        @('Phones', 200),
        @('Emails', 300)
    )
    foreach ($lbl in $($fields|select @L='Fields'; E=({$_; $fields.IndexOf($_); 0; $_ -replace '_', ' ' })))
    {
        ...
    }
    foreach ($tbx in $($fields|select @L='Fields'; E=({$_; $fields.IndexOf($_); 1; 1 })))
    {
        ...
    }
    $xBody += ...
    $bColSpan = 'Grid.ColumnSpan="7"'
    $bCol = ''
    $bRowSpan = ''
    $bRow = 'Grid.Row="1"'
    $fWidth = 280;
    $fHeight = 280;
    $fTopMost = $false;
    $bHeight = ($fHeight * 2.5);
    $bWidth = ($fWidth * 3.9);
    $objects = @(@('Empty', ''));
    $dSource = @{}
    'CsvPath', 'HtmlPath' | select @E=({$dSource.Add($_, (Invoke-Expression "$$_"))}) > $null
    $jQuery = @('Clear Highlighting', 'Clear JQuery Filter', 'Filter Using JQuery')
    $sources = @(...)
}

```

```

Process
{
    $body = (Format-Html $obj)
    $arr = @('..css/', '..js/')
    foreach ($itm in $arr)
    {
        $Head = ($Head -replace $itm, (Resolve-Path $itm))
    }
    $rDefs = '
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
    '
    $cDefs = '
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="Auto" />
    '
    $fields = @(
        @('Id', 60),
        @('Contact_Title', 80),
        @('First_Name', 120),
        @('Last_Name', 200),
        @('Phones', 200),
        @('Emails', 300)
    )
    foreach ($lbl in $($fields|select @L='Fields'; E=({$_; $fields.IndexOf($_); 0; $_ -replace '_', ' ' })))
    {
        ...
    }
    foreach ($tbx in $($fields|select @L='Fields'; E=({$_; $fields.IndexOf($_); 1; 1 })))
    {
        ...
    }
    $xBody += ...
    $bColSpan = 'Grid.ColumnSpan="7"'
    $bCol = ''
    $bRowSpan = ''
    $bRow = 'Grid.Row="1"'
    $fWidth = 280;
    $fHeight = 280;
    $fTopMost = $false;
    $bHeight = ($fHeight * 2.5);
    $bWidth = ($fWidth * 3.9);
    $objects = @(@('Empty', ''));
    $dSource = @{}
    'Server', 'Database', 'Table' | select @E=({$dSource.Add($_, (Invoke-Expression "$$_"))}) > $null
    $jQuery = @('Clear Highlighting', 'Clear JQuery Filter', 'Filter Using JQuery')
    $sources = @(...)
}

```

- As you can see, the CSV Element on the left builds its parameter list much the same way the SQL element on the right does
- Note at the top that the relative folders for css and js are resolved to full directory paths using the PowerShell Resolve-Path cmdlet. This is why it is important that the **Test Contact Type Module** sets the current working directory to being it's own location \Documents\Source Files\ps1\ so that relative paths like '..css/' and '..js/' will be resolved to \Documents\Source Files\css\ and \Documents\Source Files\js\
- The Format-Html function is from the ScriptBlocks module
  - Which uses PowerShell's ConvertTo-Html cmdlet to convert the data-table object
  - Then takes the table element from that object and reformats it by repaginating the elements within
- The rDefs and cDefs get parsed by PresentationFramework when they finally arrive in the FormWPFWebView module
- The \$fields array is used to build the labels and textboxes

```

foreach ($lbl in $($fields|select @L='Fields'; E=({$_; $fields.IndexOf($_); 0; $_ -replace '_', ' ' })))
{
    $xBody += "
    <Label
    Name = ``$lbl.Fields[0]``
    Width = ``$($lbl.Fields[1])``
    Grid.Column = ``$($lbl.Fields[2])``
    Grid.Row = ``$($lbl.Fields[3])``
    Content = ``$($lbl.Fields[4])``
    "
    />
}

```

```

foreach ($tbx in $($fields|select @L='Fields'; E=({$_; $fields.IndexOf($_); 1; 1 })))
{
    $xBody += "
    <TextBox
    Name = ``$($tbx.Fields[0])``
    Width = ``$($tbx.Fields[1])``
    Grid.Column = ``$($tbx.Fields[2])``
    Grid.Row = ``$($tbx.Fields[3])``
    Margin = ``$($tbx.Fields[4])``
    "
    />
}

```

```

1362 $fields = @(...)
1370 foreach ($lbl in $($fields|select @{L='Fields'; E=({$_; $fields.IndexOf($_); 0; $_ -replace '_', ' '}})))
1371 {
1372     $xBody += "
1373     <Label
1374         Name = ``"lbl$($lbl.Fields[0])``"
1375         Width = ``$($lbl.Fields[1])``"
1376         Grid.Column = ``$($lbl.Fields[2])``"
1377         Grid.Row = ``$($lbl.Fields[3])``"
1378         Content = ``$($lbl.Fields[4])``"
1379     />
1380 "
1381 }
1382 foreach ($tbx in $($fields|select @{L='Fields'; E=({$_; $fields.IndexOf($_); 1; 1}})))
1383 {
1384     $xBody += "
1385     <TextBox
1386         Name = ``$($tbx.Fields[0])``"
1387         Width = ``$($tbx.Fields[1])``"
1388         Grid.Column = ``$($tbx.Fields[2])``"
1389         Grid.Row = ``$($tbx.Fields[3])``"
1390         Margin = ``$($tbx.Fields[4])``"
1391     />
1392 "
1393 }
1394 $xBody += '
1395 <ComboBox
1396     Name = "cboUpdate"
1397     Grid.Column = "6"
1398     Grid.Row = "1"
1399     Margin = "1"
1400 />
1401 '

```

- After the labels and textboxes are built, a combo box is added. This combo box will be used to list functions to call from the WPF Form

```

$objects = @(@('Empty', ''));
$dSource = @{}
'CsvPath', 'HtmlPath' | select @{E=({$dSource.Add($_, (Invoke-Expression "`$_"))}} > $null
$jQuery = @('Clear Highlighting', 'Clear JQuery Filter', 'Filter Using JQuery')

$dSource = @{}
'Server', 'Database', 'Table' | select @{E=({$dSource.Add($_, (Invoke-Expression "`$_"))}} > $null

```

- As I am using a combo box to call functions I do not need buttons. To use buttons, I would fill the \$objects arrays with button names and the functions they called or text boxes they should get data from like I do in the proof of concept sample

```

if ($objects.Count -eq 0) {
    $objects = @(@('btnUpdate', 'txt1'), @('btnUpdate1', 'txt1'), @())
}
if ($dSource.Count -eq 0) {

```

- \$dSource is an important parameter set as it passes the file locations or SQL Database and table names back and forth with function calls
- \$jQuery is also important, as it tells the WPF form to skip running other functions.

```

$sources = @(
    @('cboUpdate',
        @('Select a Function', 'Add Contact', 'Clear Fields', 'Clear Highlighting', 'Clear JQuery Filter', 'Delete Contact', 'Get All Contacts',
            'Filter Using JQuery', 'Filter Using OLEDB', 'Send to Email New', 'Send to Email Open', 'Send to PowerPoint New', 'Send to PowerPoint Open',
            'Send to Word New', 'Send to Word Open', 'Update Contact'),
        @('Id', 'Contact_Title', 'First_Name', 'Last_Name', 'Phones', 'Emails'),
        @(
            @(''),
            @('Set-Contact', 'Insert-ContactCSV', 'Show-ContactCSV'),
            @(''),
            @('clearHighlight'),
            @('clearFilter'),
            @('Delete-ContactCSV', 'Show-ContactCSV'),
            @('Show-ContactCSV'),
            @('filterRows'),
            @('Show-ContactCSV'),
            @('Send-ContactsEmailNew'),
            @('Send-ContactsEmail'),
            @('Send-ContactsPowerPointNew'),
            @('Send-ContactsPowerPointOpen'),
            @('Send-ContactsWordNew'),
            @('Send-ContactsWordOpen'),
            @('Set-Contact', 'Update-ContactCSV', 'Show-ContactCSV' )
        )
    ),
    @()
)

$sources = @(
    @('cboUpdate',
        @('Select a Function', 'Add Contact', 'Clear Fields', 'Clear Highlighting', 'Clear JQuery Filter', 'Delete Contact', 'Get All Contacts',
            'Filter Using JQuery', 'Filter Using SQL', 'Send to Email New', 'Send to Email Open', 'Send to PowerPoint New', 'Send to PowerPoint Open',
            'Send to Word New', 'Send to Word Open', 'Update Contact'),
        @('Id', 'Contact_Title', 'First_Name', 'Last_Name', 'Phones', 'Emails'),
        @(
            @(''),
            @('Set-Contact', 'Insert-ContactSQL', 'Show-ContactSQL'),
            @(''),
            @('clearHighlight'),
            @('clearFilter'),
            @('Delete-ContactSQL', 'Show-ContactSQL'),
            @('Show-ContactSQL'),
            @('filterRows'),
            @('Show-ContactSQL'),
            @('Send-ContactsEmailNew'),
            @('Send-ContactsEmail'),
            @('Send-ContactsPowerPointNew'),
            @('Send-ContactsPowerPointOpen'),
            @('Send-ContactsWordNew'),
            @('Send-ContactsWordOpen'),
            @('Set-Contact', 'Update-ContactSQL', 'Show-ContactSQL' )
        )
    ),
    @()
)

```

- As you can see, both \$sources parameters are almost identical
  - The cboUpdate array is used
    - In its 1<sup>st</sup> array to populate the combo box used in making function calls
    - In its 2<sup>nd</sup> array to identify the fields it and the WebBrowser control will be interacting with
    - And, in it's 3<sup>rd</sup> array, gives the names of the functions to call when an item in the combo box is selected

```

End
{
    if (!$HtmlOnly)
    {
        $prm = @{}
        'head', 'title', 'body', 'cDefs', 'xBody', 'bColSpan', 'objects', 'fWidth', 'fHeight', 'fTopMost', 'sources', 'dSource', 'jQuery' | select @E=({$prm.Add($_, (Invoke-Expression "$$_"))}) > $null
        Show-WPFWebViewForm -object (Set-WebForm @prm)
    }
    else
    {
        return ((
            ConvertTo-Html -Body $body -Head $Head -Title $Title) -replace '<html xmlns="http://www.w3.org/1999/xhtml">',
            '<html xmlns="http://www.w3.org/1999/xhtml"><meta http-equiv="x-ua-compatible" content="IE=11">'
        ))
    }
}

```

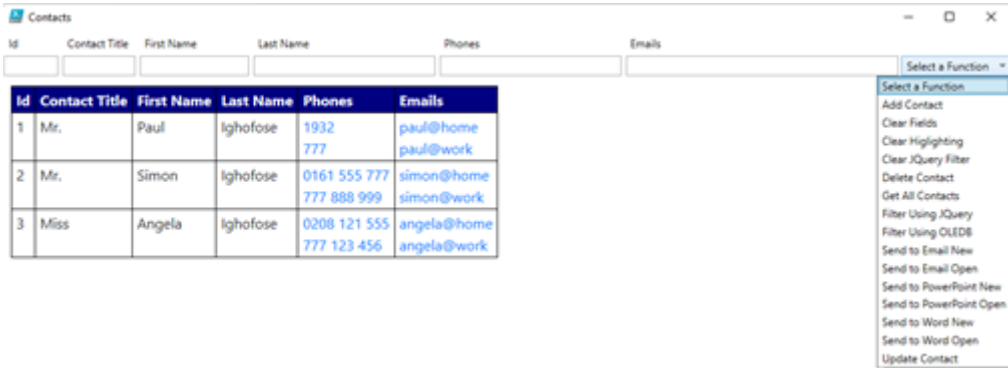
- Finally, and this is identical for both CSV and SQL
  - if calling the function from the WPF Form the parameter \$HtmlOnly is set to true and just the WebBrowser control is updated
  - otherwise, the all parameters are put in to a hash table and passed to the Set-WebForm function of the TypeWebForm module
    - Which then wraps them all up in a Class Object
  - Then the returned Class Object is sent to the Show-WPFWebViewForm function of the FormWPFWebView module
    - Which then displays the results

```

1  Add-Type -AssemblyName PresentationFramework
2
3  function Show-WPFWebViewForm {
4      [CmdletBinding()]
5      Param (
6          [...]
10         [ValidateNotNullOrEmpty()]
11         [psobject]
12         $object
13     )
14     $sinnerHTML = ... + $object.Title + '
21     </title>
22     <script>
23         + $object.Script + ... + $object.Style + '
27     </style>
28     + $object.Head + ... + $object.Body + '
32     </body>
33     </html>
34
35     [xml]$XAML = ... + $object.Title + ''
40     WindowStartupLocation=" + $object.WindowLocation + ... + $object.WindowSize + ''
42     Top=" + $object.WindowTop + ... + $object.RowDefinitions + '
46     </Grid.RowDefinitions>
47     <Grid.ColumnDefinitions>
48         + $object.ColumnDefinitions + ... + $object.XAMLBody + '
51     <WebBrowser
52         HorizontalAlignment="Left"
53         Margin="10,10,10,10"
54         VerticalAlignment="Top"
55         Name="WebBrowser"
56         + $object.BrowserColSpan + ... + $object.BrowserColumn + '
58         + $object.BrowserRowSpan + ... + $object.BrowserRow + '
60     />
61 </Grid>
62 </Window>
63
64 $reader = New-Object System.Xml.XmlNodeReader($XAML)
65 $_ = ConvertFrom-Json "{LoadIt: 0}"
66 $form = [Windows.Markup.XamlReader]::Load($reader)
67 $form.Width = $object.FormWidth
68 $form.Topmost = $object.FormTopMost
69 $webBrowser = $form.FindName('WebBrowser')
70 $webBrowser.Width = $object.BrowserWidth
71 $webBrowser.Height = $object.BrowserHeight
72 $webBrowser.NavigateToString($sinnerHTML)
73 $webBrowser.Add_Navigated(...)
90
91 if ($object.Sources[0] -ne 'Empty') {...}
184
185 if ($object.Objects[0] -ne 'Empty') {...}
200 $form.ShowDialog()
201 }

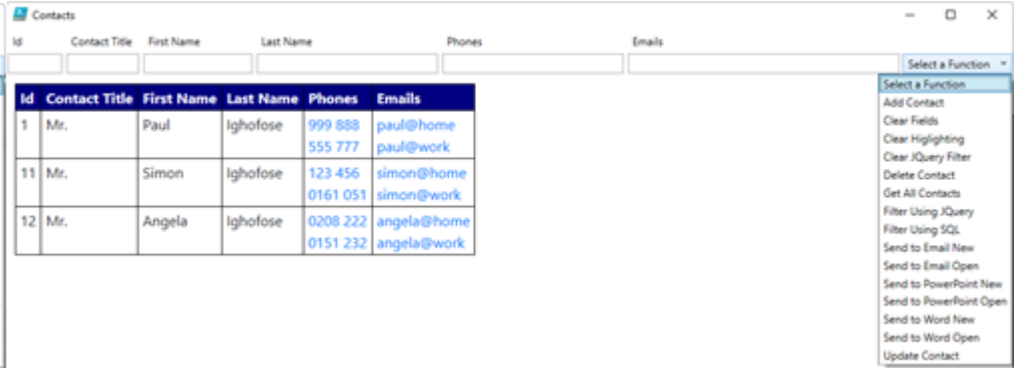
```

## WPF Web Form View Results

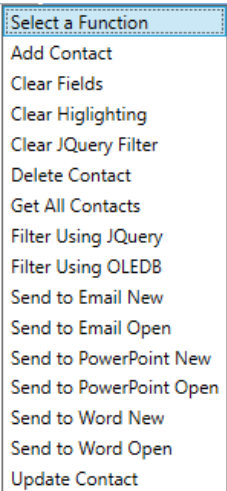


The screenshot shows a web form titled 'Contacts' with a table containing 3 rows of contact data. A dropdown menu 'Select a Function' is open, displaying a list of actions including 'Add Contact', 'Clear Fields', 'Clear Highlighting', 'Clear JQuery Filter', 'Delete Contact', 'Get All Contacts', 'Filter Using JQuery', 'Filter Using OLEDB', 'Send to Email New', 'Send to Email Open', 'Send to PowerPoint New', 'Send to PowerPoint Open', 'Send to Word New', 'Send to Word Open', and 'Update Contact'.

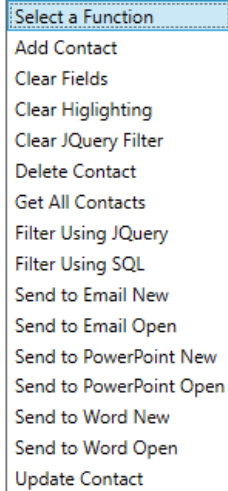
Id	Contact Title	First Name	Last Name	Phones	Emails
1	Mr.	Paul	Ighofose	1932 777	paul@home paul@work
2	Mr.	Simon	Ighofose	0161 555 777 777 888 999	simon@home simon@work
3	Miss	Angela	Ighofose	0208 121 555 777 123 456	angela@home angela@work



This screenshot is identical to the one on the left, showing the same 'Contacts' web form with 3 contacts and the 'Select a Function' dropdown menu open.




A close-up view of the 'Select a Function' dropdown menu, showing the list of actions: Add Contact, Clear Fields, Clear Highlighting, Clear JQuery Filter, Delete Contact, Get All Contacts, Filter Using JQuery, Filter Using OLEDB, Send to Email New, Send to Email Open, Send to PowerPoint New, Send to PowerPoint Open, Send to Word New, Send to Word Open, and Update Contact.



Another close-up view of the 'Select a Function' dropdown menu, identical to the one on the left.


- If you do use Bootstrap css like I have, ensure that you do not set the media=screen in the <link></link> tags, as this will cause the JQuery function I have provided to fail.
- The css file I have provided should have its media=screen in the <link></link> tags, as the JQuery function I have provided uses it to create a style tag in the response it sends back to the WPF form for use in formatting html sent to Outlook items
- *Clear Fields*, will clear all entries in the Form Text boxes
- *Clear Highlighting* will use both JavaScript and JQuery to clear all cells highlighted by the JQuery on-click function
- *Clear JQuery* filter will unhide all rows hidden by the *Filter Using JQuery* function. Hidden rows are not sent to Outlook, PowerPoint or Word.
- *Filter Using OLEDB* or *SQL* will limit the table result after querying their relative data sources
- *Send To* functions will create hyperlinks in Outlook, PowerPoint and Word where they exist. Also note that the hyperlinks displayed are created dynamically using JQuery in the JavaScript file provided, and arrays are split on to separate lines

 Contacts

Id	Contact Title	First Name	Last Name	Phones	Emails
1	Mr.	Paul			paul@home, paul@work

Id	Contact Title	First Name	Last Name	Phones	Emails
1	Mr.	Paul	Ighofose	1932 777	paul@home paul@work
2	Mr.	Simon	Ighofose	0161 555 777 777 888 999	simon@home simon@work
3	Miss	Angela	Ighofose	0208 121 555 777 123 456	angela@home angela@work

 Contacts

Id	Contact Title	First Name	Last Name	Phones	Emails
	Mr.	Paul			paul@home, paul@work

Id	Contact Title	First Name	Last Name	Phones	Emails
1	Mr.	Paul	Ighofose	1932 777	paul@home paul@work
2	Mr.	Simon	Ighofose	0161 555 777 777 888 999	simon@home simon@work
3	Miss	Angela	Ighofose	0208 121 555 777 123 456	angela@home angela@work

- Clicking the cells will
  - either highlight or remove highlighting
  - populate the textboxes or remove entries from those textboxes
  - convert separated line entries back in to comma separated arrays
  - when working with just one row and having clicked all its cells, you can then update parts of that contact and run the *Update Contact* function



Contacts

Id	Contact Title	First Name	Last Name	Phones	Emails
1		Simon			

Id	Contact Title	First Name	Last Name	Phones	Emails
1	Mr.	Paul	Ighofose	1932 777	paul@home paul@work
2	Mr.	Simon	Ighofose	0161 555 777 777 888 999	simon@home simon@work

Contacts

Id	Contact Title	First Name	Last Name	Phones	Emails
1  3					

Id	Contact Title	First Name	Last Name	Phones	Emails
1	Mr.	Paul	Ighofose	1932 777	paul@home paul@work
3	Miss	Angela	Ighofose	0208 121 555 777 123 456	angela@home angela@work


- Using the *Filter Using JQuery* function will
  - Hide all rows that do not have a cell highlighted.
  - All rows hidden will not be sent when sending the table to Outlook, PowerPoint or Word
  - Although this form of filtering may allow records to be displayed when using JQuery, they may cause no records to be returned when using *Filter Using OLEDB* or *SQL*
  - Multiple entries of filtered rows are separated with a | (bar/pipe symbol)
    - When the Id field is populated with more than one entry, then only the *Delete Contact* function will use those values in its filter
- Using the *Clear JQuery Filter* will redisplay all rows and leave highlighting as it is

Contacts

Id	Contact Title	First Name	Last Name	Phones	Emails
	Mr.	Paul	Ighofose	999 888, 555 777	paul@home, paul@work

Id	Contact Title	First Name	Last Name	Phones	Emails
1	Mr.	Paul	Ighofose	999 888 555 777	paul@home paul@work
11	Mr.	Simon	Ighofose	123 456 0161 051	simon@home simon@work
12	Mr.	Angela	Ighofose	0208 222 0151 232	angela@home angela@work

**Duplicate Contact Exists**

 A contact with First Name 'Paul' and Last Name 'Ighofose' already exists!

Do you want to continue and add the record?

Yes No

- Trying to add a Contact that already exists will throw a PowerShell error warning and give you the option to continue.



```

PowerShell 1 PowerShell 4 X
+
+ CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
+ FullyQualifiedErrorId : ArgumentException

WARNING: A contact with First Name 'Paul' and Last Name 'Ighofose' already exists!
Add Contact: The running command stopped because the user selected the Stop option.
WARNING: A contact with First Name 'Paul' and Last Name 'Ighofose' already exists!
Add Contact: The running command stopped because the user selected the Stop option.

```

- Although the console can be hidden when running from a desktop link, this send the results to the PowerShell console

Contacts

Id	Contact Title	First Name	Last Name	Phones	Emails
18					

Id	Contact Title	First Name	Last Name	Phones	Emails
1	Mr.	Paul	Ighofose	999 888 555 777	paul@home paul@work
11	Mr.	Simon	Ighofose	123 456 0161 051	simon@home simon@work
12	Mr.	Angela	Ighofose	0208 222 0151 232	angela@home angela@work
18	Mr.	Paul	Ighofose	999 888 555 777	paul@home paul@work

Delete Contact Confirmation

Contacts with Id: '18' will be deleted!  
Do you want to continue?

Yes No

- The same will happen when deleting contacts, an error warning will request confirmation before deleting those contacts

**Contacts**

Id	Contact Title	First Name	Last Name	Phones	Emails
18	Mr.	Paulo	Nutini	888 55 22, 33 11 05	paulo@home, paulo@work

Id	Contact Title	First Name	Last Name	Phones	Emails
1	Mr.	Paul	Ighofose	999 888 555 777	paul@home paul@work
11	Mr.	Simon	Ighofose	123 456 0161 051	simon@home simon@work
12	Mr.	Angela	Ighofose	0208 222 0151 232	angela@home angela@work
18	Mr.	Paulo	Nutini	888 55 22 33 11 05	paulo@home paulo@work

- Instead of deleting the Contact you can just use the *Update Contact* function after changing the entries to update the contact with the given Id

Untitled - Message (HTML)

FILE MESSAGE INSERT OPTIONS FORMAT TEXT REVIEW DEVELOPER

DESIGN LAYOUT

Cut Copy Paste Format Painter Clipboard

Arial 11 A A Basic Text

Address Book Names Check Names Attach File Attach Item Signature Include

Follow Up High Importance Low Importance Tags Zoom

To... Cc... Subject

Send

Id	Contact Title	First Name	Last Name	Phones	Emails
1	Mr.	Paul	Ighofose	999 888 555 777	paul@home paul@work
11	Mr.	Simon	Ighofose	123 456 0161 051	simon@home simon@work
12	Mr.	Angela	Ighofose	0208 222 0151 232	angela@home angela@work
18	Mr.	Paulo	Nutini	888 55 22 33 11 05	paulo@home paulo@work

tel:999 888  
Ctrl+Click to follow link

- Using either the *Send to Email New* or *Open* functions will send the table displayed, its formatting and any hyperlinks to an Outlook item

Presentation1 - PowerPoint

FILE HOME INSERT DESIGN TRANSITIONS ANIMATIONS SLIDE SHOW REVIEW VIEW DEVELOPER FOXIT PDF

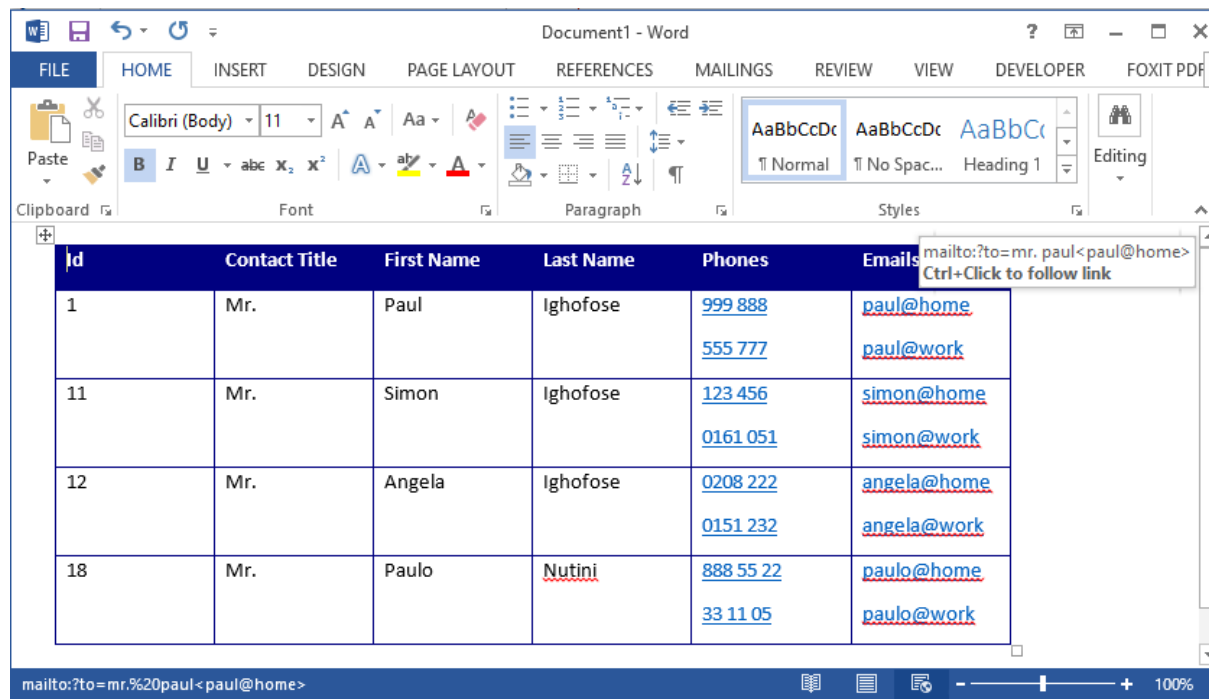
1

Id	Contact Title	First Name	Last Name	Phones	Emails
1	Mr.	Paul	Ighofose	<a href="tel:999888555777">999 888 555 777</a>	<a href="mailto:paul@home">paul@home</a> <a href="mailto:paul@work">paul@work</a>
11	Mr.	Simon	Ighofose	<a href="tel:1234560161051">123 456 0161 051</a>	<a href="mailto:simon@home">simon@home</a> <a href="mailto:simon@work">simon@work</a>
12	Mr.	Angela	Ighofose	<a href="tel:02082220151232">0208 222 0151 232</a>	<a href="mailto:angela@home">angela@home</a> <a href="mailto:angela@work">angela@work</a>
18	Mr.	Paulo	Nutini	<a href="tel:8885522331105">888 55 22 33 11 05</a>	<a href="mailto:paulo@home">paulo@home</a> <a href="mailto:paulo@work">paulo@work</a>

Id	Contact Title	First Name	Last Name	Phones	Emails
1	Mr.	Paul	Ighofose	<a href="tel:999888555777">999 888 555 777</a>	<a href="mailto:paul@home">paul@home</a> <a href="mailto:paul@work">paul@work</a>
11	Mr.	Simon	Ighofose	<a href="tel:1234560161051">123 456 0161 051</a>	<a href="mailto:simon@home">simon@home</a> <a href="mailto:simon@work">simon@work</a>
12	Mr.	Angela	Ighofose	<a href="tel:02082220151232">0208 222 0151 232</a>	<a href="mailto:angela@home">angela@home</a> <a href="mailto:angela@work">angela@work</a>
18	Mr.	Paulo	Nutini	<a href="tel:8885522331105">888 55 22 33 11 05</a>	<a href="mailto:paulo@home">paulo@home</a> <a href="mailto:paulo@work">paulo@work</a>

<mailto:Mr.%20Paulo%20Nutini>

- Using either the *Send to PowerPoint New* or *Open* functions will send the table displayed, its formatting and any hyperlinks to a PowerPoint presentation, but with PowerPoint, hyperlinks are only active in Slide Show mode.
- If the table has more rows than can fit on one slide, then new slides are created with row headers
- The Sending to Open function inserts a new slide below the current slide on display and before the next slide if there is one below it



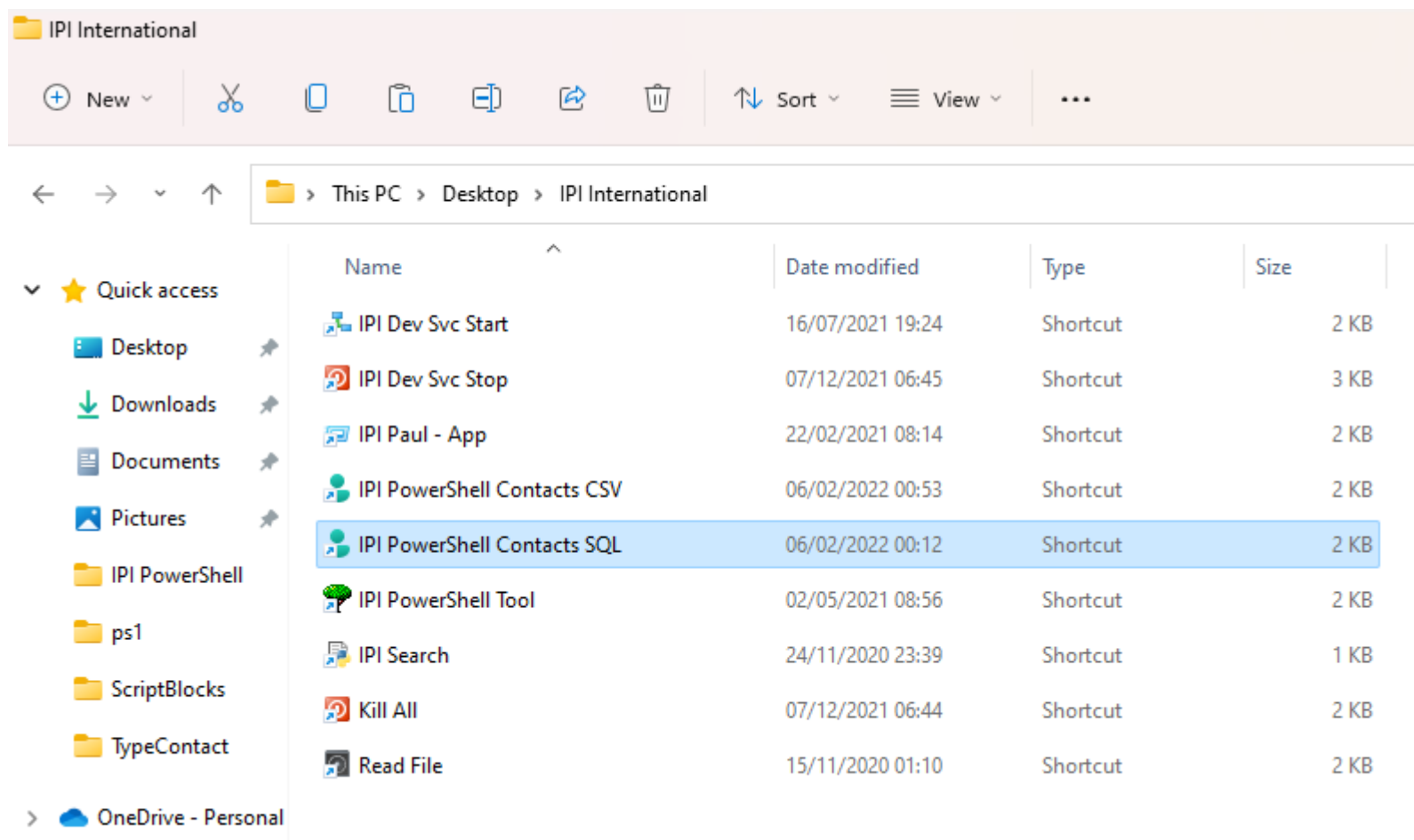
- Using either the *Send to Word New* or *Open* functions will send the table displayed, its formatting and any hyperlinks to a Word Document

Id	Contact Title	First Name	Last Name	Phones	Emails
1	Mr.	Paul	Ighofose	1932 777	paul@home paul@work

Id	Contact Title	First Name	Last Name	Phones	Emails
2	Mr.	Simon	Ighofose	0161 555 777 777 888 999	simon@home simon@work
3	Miss	Angela	Ighofose	0208 121 555 777 123 456	angela@home angela@work

- Like PowerPoint, if the table cannot fit on one page, then it continues on to a new page and the header is repeated on that page



- You can create a desktop link for the Contacts CSV version and name it IPI PowerShell Contacts CSV
- Then use the path below to run the Contact script and load the WPF Web View Form (best done after you have created your 1<sup>st</sup> contact)

```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -ExecutionPolicy Bypass -WindowStyle Hidden -File
"%userprofile%\Documents\Source Files\ps1\Test Contact Type Module.ps1" -ShowCSV 1
```

- And likewise for IPI PowerShell Contacts SQL

```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -ExecutionPolicy Bypass -WindowStyle Hidden -File
"%userprofile%\Documents\Source Files\ps1\Test Contact Type Module.ps1" -ShowSql 1
```

**Excuse me if any changes I made whilst creating this help file leads to a function not working  
(Late breaking changes always do that ☹ )**