```
NAME
    Export-Excel

SYNOPSIS
    Exports data to an Excel worksheet.


    -------------------------- EXAMPLE 1 --------------------------

    PS C:\>Get-Process | Export-Excel .\Test.xlsx -show

    Export all the processes to the Excel file 'Test.xlsx' and open the file immedi
ately.




    -------------------------- EXAMPLE 2 --------------------------

    >

    PS> $ExcelParams = @{
            Path    = $env:TEMP + '\Excel.xlsx'
            Show    = $true
            Verbose = $true
        }
        Remove-Item -Path $ExcelParams.Path -Force -EA Ignore
        Write-Output -1 668 34 777 860 -0.5 119 -0.1 234 788 |
            Export-Excel @ExcelParams -NumberFormat '[Blue]$#,##0.00;[Red]-$#,##0.0
0'

        Exports all data to the Excel file 'Excel.xslx' and colors the negative val
ues
        in Red and the positive values in Blue. It will also add a dollar sign in f
ront
        of the numbers which use a thousand seperator and display to two decimal pl
aces.




    -------------------------- EXAMPLE 3 --------------------------

    >

    PS> $ExcelParams = @{
            Path    = $env:TEMP + '\Excel.xlsx'
            Show    = $true
            Verbose = $true
        }
        Remove-Item -Path $ExcelParams.Path -Force -EA Ignore
        [PSCustOmobject][Ordered]@{
            Date      = Get-Date
            Formula1  = '=SUM(F2:G2)'
            String1   = 'My String'
            String2   = 'a'
            IPAddress = '10.10.25.5'
            Number1   = '07670'
            Number2   = '0,26'
            Number3   = '1.555,83'
            Number4   = '1.2'
            Number5   = '-31'
            PhoneNr1  = '+32 44'
            PhoneNr2  = '+32 4 4444 444'
            PhoneNr3  =  '+3244444444'
        } | Export-Excel @ExcelParams -NoNumberConversion IPAddress, Number1

        Exports all data to the Excel file "Excel.xlsx" and tries to convert all va
lues
        to numbers where possible except for "IPAddress" and "Number1", which are
        stored in the sheet 'as is', without being converted to a number.




    -------------------------- EXAMPLE 4 --------------------------
    >

    PS> $ExcelParams = @{
            Path    = $env:TEMP + '\Excel.xlsx'
            Show    = $true
            Verbose = $true
        }
        Remove-Item -Path $ExcelParams.Path -Force -EA Ignore
        [PSCustOmobject][Ordered]@{
            Date      = Get-Date
            Formula1  = '=SUM(F2:G2)'
            String1   = 'My String'
            String2   = 'a'
            IPAddress = '10.10.25.5'
            Number1   = '07670'
            Number2   = '0,26'
            Number3   = '1.555,83'
            Number4   = '1.2'
            Number5   = '-31'
            PhoneNr1  = '+32 44'
            PhoneNr2  = '+32 4 4444 444'
            PhoneNr3  =  '+3244444444'
        } | Export-Excel @ExcelParams -NoNumberConversion *

        Exports all data to the Excel file 'Excel.xslx' as is, no number conversion
        will take place. This means that Excel will show the exact same data that
        you handed over to the 'Export-Excel' function.




    -------------------------- EXAMPLE 5 --------------------------

    >

    PS> $ExcelParams = @{
            Path    = $env:TEMP + '\Excel.xlsx'
            Show    = $true
            Verbose = $true
        }
        Remove-Item -Path $ExcelParams.Path -Force -EA Ignore
        Write-Output 489 668 299 777 860 151 119 497 234 788 |
            Export-Excel @ExcelParams -ConditionalText $(
                New-ConditionalText -ConditionalType GreaterThan 525 -ConditionalTe
xtColor DarkRed -BackgroundColor LightPink
            )

        Exports data that will have a Conditional Formatting rule in Excel
        that will show cells with a value is greater than 525, whith a
        background fill color of "LightPink" and the text in "DarkRed".
        Where condition is not met the color willbe the default, black
        text on a white background.




    -------------------------- EXAMPLE 6 --------------------------

    >

    PS> $ExcelParams = @{
            Path    = $env:TEMP + '\Excel.xlsx'
            Show    = $true
            Verbose = $true
        }
        Remove-Item -Path $ExcelParams.Path -Force -EA Ignore
        Get-Service | Select-Object -Property Name, Status, DisplayName, ServiceNam
e |
            Export-Excel @ExcelParams -ConditionalText $(
                New-ConditionalText Stop DarkRed LightPink
                New-ConditionalText Running Blue Cyan
            )

        Exports all services to an Excel sheet, setting a Conditional formatting ru
le
        that will set the background fill color to "LightPink" and the text color
        to "DarkRed" when the value contains the word "Stop".
        If the value contains the word "Running" it will have a background fill
        color of "Cyan" and text colored 'Blue'. If neither condition is met, the
        color will be the default, black text on a white background.




    -------------------------- EXAMPLE 7 --------------------------

    >

    PS> $ExcelParams = @{
            Path      = $env:TEMP + '\Excel.xlsx'
            Show      = $true
            Verbose   = $true
        }
        Remove-Item -Path $ExcelParams.Path -Force -EA Ignore

        $Array = @()

        $Obj1 = [PSCustomObject]@{
            Member1   = 'First'
            Member2   = 'Second'
        }

        $Obj2 = [PSCustomObject]@{
            Member1   = 'First'
            Member2   = 'Second'
            Member3   = 'Third'
        }

        $Obj3 = [PSCustomObject]@{
            Member1   = 'First'
            Member2   = 'Second'
            Member3   = 'Third'
            Member4   = 'Fourth'
        }

        $Array = $Obj1, $Obj2, $Obj3
        $Array | Out-GridView -Title 'Not showing Member3 and Member4'
        $Array | Update-FirstObjectProperties | Export-Excel @ExcelParams -Workshee
tName Numbers

        Updates the first object of the array by adding property 'Member3' and 'Mem
ber4'.
        Afterwards. all objects are exported to an Excel file and all column header
s are visible.




    -------------------------- EXAMPLE 8 --------------------------

    PS C:\>Get-Process | Export-Excel .\test.xlsx -WorksheetName Processes -Include
PivotTable -Show -PivotRows Company -PivotData PM






    -------------------------- EXAMPLE 9 --------------------------

    PS C:\>Get-Process | Export-Excel .\test.xlsx -WorksheetName Processes -ChartTy
pe PieExploded3D -IncludePivotChart -IncludePivotTable -Show
    -PivotRows Company -PivotData PM






    -------------------------- EXAMPLE 10 --------------------------

    PS C:\>Get-Service | Export-Excel 'c:\temp\test.xlsx'  -Show -IncludePivotTable
 -PivotRows status -PivotData @{status='count'}
```

```
    -------------------------- EXAMPLE 11 --------------------------

    >

    PS> $pt = [ordered]@{}
        $pt.pt1=@{ SourceWorkSheet   = 'Sheet1';
                   PivotRows         = 'Status'
                   PivotData         = @{'Status'='count'}
                   IncludePivotChart = $true
                   ChartType         = 'BarClustered3D'
        }
        $pt.pt2=@{ SourceWorkSheet   = 'Sheet2';
                   PivotRows         = 'Company'
                   PivotData         = @{'Company'='count'}
                   IncludePivotChart = $true
                   ChartType         = 'PieExploded3D'
        }
        Remove-Item  -Path .\test.xlsx
        Get-Service | Select-Object    -Property Status,Name,DisplayName,StartType
| Export-Excel -Path .\test.xlsx -AutoSize
        Get-Process | Select-Object    -Property Name,Company,Handles,CPU,VM
| Export-Excel -Path .\test.xlsx -AutoSize -WorksheetName 'sheet2'
        Export-Excel -Path .\test.xlsx -PivotTableDefinition $pt -Show

        This example defines two PivotTables. Then it puts Service data on Sheet1
        with one call to Export-Excel and Process Data on sheet2 with a second
        call to Export-Excel. The third and final call adds the two PivotTables
        and opens the spreadsheet in Excel.




    -------------------------- EXAMPLE 12 --------------------------

    >

    PS> Remove-Item  -Path .\test.xlsx
        $excel = Get-Service | Select-Object -Property Status,Name,DisplayName,Star
tType | Export-Excel -Path .\test.xlsx -PassThru
        $excel.Workbook.Worksheets["Sheet1"].Row(1).style.font.bold = $true
        $excel.Workbook.Worksheets["Sheet1"].Column(3 ).width = 29
        $excel.Workbook.Worksheets["Sheet1"].Column(3 ).Style.wraptext = $true
        $excel.Save()
        $excel.Dispose()
        Start-Process .\test.xlsx

        This example uses -PassThru. It puts service information into sheet1 of the
        workbook and saves the ExcelPackage object in $Excel. It then uses the pack
age
        object to apply formatting, and then saves the workbook and disposes of the
 object
        before loading the document in Excel. Other commands in the module remove t
he need
        to work directly with the package object in this way.




    -------------------------- EXAMPLE 13 --------------------------

    >

    PS> Remove-Item -Path .\test.xlsx -ErrorAction Ignore
        $excel = Get-Process | Select-Object -Property Name,Company,Handles,CPU,PM,
NPM,WS | Export-Excel -Path .\test.xlsx -ClearSheet -WorksheetName
    "Processes" -PassThru
        $sheet = $excel.Workbook.Worksheets["Processes"]
        $sheet.Column(1) | Set-ExcelRange -Bold -AutoFit
        $sheet.Column(2) | Set-ExcelRange -Width 29 -WrapText
        $sheet.Column(3) | Set-ExcelRange -HorizontalAlignment Right -NFormat "#,##
#"
```

```
        Set-ExcelRange -Address $sheet.Cells["E1:H1048576"]  -HorizontalAlignment R
ight -NFormat "#,###"
        Set-ExcelRange -Address $sheet.Column(4)  -HorizontalAlignment Right -NForm
at "#,##0.0" -Bold
        Set-ExcelRange -Address $sheet.Row(1) -Bold -HorizontalAlignment Center
        Add-ConditionalFormatting -WorkSheet $sheet -Range "D2:D1048576" -DataBarCo
lor Red
        Add-ConditionalFormatting -WorkSheet $sheet -Range "G2:G1048576" -RuleType
GreaterThan -ConditionValue "104857600" -ForeGroundColor Red
        foreach ($c in 5..9) {Set-ExcelRange -Address $sheet.Column($c)  -AutoFit }
        Export-Excel -ExcelPackage $excel -WorksheetName "Processes" -IncludePivotC
hart -ChartType ColumnClustered -NoLegend -PivotRows company
    -PivotData @{'Name'='Count'}  -Show

        This a more sophisticated version of the previous example showing different
        ways of using Set-ExcelRange, and also adding conditional formatting.
        In the final command a PivotChart is added and the workbook is opened in Ex
cel.




    -------------------------- EXAMPLE 14 --------------------------

    PS C:\>0..360 | ForEach-Object {[pscustomobject][ordered]@{X=$_; Sinx="=Sin(Rad
ians(x)) "} } | Export-Excel -now -LineChart -AutoNameRange

    Creates a line chart showing the value of Sine(x) for values of X between 0 and
 360 degrees.




    -------------------------- EXAMPLE 15 --------------------------

    >

    PS> Invoke-Sqlcmd -ServerInstance localhost\DEFAULT -Database AdventureWorks201
4 -Query "select *  from sys.tables" -OutputAs DataRows |
        Export-Excel -Path .\SysTables_AdventureWorks2014.xlsx -WorksheetName Table
s

        Runs a query against a SQL Server database and outputs the resulting rows D
ataRows using the -OutputAs parameter.
        The results are then piped to the Export-Excel function.
        NOTE: You need to install the SqlServer module from the PowerShell Gallery
in oder to get the -OutputAs parameter for the Invoke-Sqlcmd cmdlet.
```