PowerShell について

e.wakabayashi-aa

2019 年 8 月 30 日

概要

DOS のバッチコマンドについて調べていたときに知ったのが PowerShell です。コマンドプロンプト (cmd.exe) の後継らしきものであることは分かったもののコマンドは冗長でオプションは意味不明。情報は少なくとくに初心者向けのものはほとんど見つかりません。この状況はいまもあまり変わっていません。PowerShell は管理者向けに作られたものであり、.NET(ドットネット) は開発者の使うものであり、そもそも初心者が使うことを想定していないのです。5000 円もした分厚い解説書は半年間埃をかぶるままでしたが必要にせまられ少しづつ解明していきました。データを構造化できる pscustomobject と外部からエクセル VBA を操作する仕組み interop を知ってから手放せないもになりました。ほとんどすべてのことが PowerShell でできます。すべての情報はネット上にあります。しかしその勘所は暗黙知です。

目次

1　PowerShell とは

マイクロソフトが開発したコマンド・ライン・インターフェース (CLI) シェル、およびスクリプト言語。一言でいえば DOS プロンプト（DOS 窓）の後継ですが機能はそれをはるかに上回ります。バッチを走らせることができるだけでなく開発者から見た Windows ともいえる **.NET Framework** を使えるのでおよそ Windows できることはすべてできます。**Microsoft.Office.Interop.Excel** を介せば外部からエクセル・マクロ（VB) と同等のコマンドでエクセル・ファイルをセル単位で操作することが可能です。

2　どこにある

Windows 標準搭載 (Windows7〜) なのでダウンロードしたりインストールしたりする必要はない。インストール済みなのでスタート・メニューを開けば出てくるのだが PC によってその場所は微妙に異なる。ちなみに現在私が使用中の PC(Windows10) ではそのもずばり「Windows PowerShell」フォルダとして「W」項目内で見つけることがえきる。またフォルダ・メニューの「ファイル」に「Windows PowerShell を開く」項目がありそこから実行することができる。本体 (実行ファイル) は `C:\WINDOWS\system32\WindowsPowerShell\v1.0` にある。アプリケーションとしては変わった場所だが DOS(cmd.exe) の後継と考えればそれも納得がいく。

2.1　ISE か否か

ただの **Windows PowerShell** と **Windows PowerShell ISE** の 2 種類がある。ISE とは「統合開発環境」のこと。ISE 無しは DOS 窓の後継としての意味しかない（ように思える）のでもっぱら使うのは ISE のほうを、拡張子「.ps1」の関連づけも ISE のほうにしています。

2.2　32bit か 64bit か

末尾に **(x86)** の付いたのが 32bit、付いてないのが 64bit。基本的にどちらを使っても同じですが 64bit マシンなので 64bit を使っています。この違いが問題になるのはデータ・ベースへの接続を行うときです。ODBC ドライバには 32bit 用しかないものがあります。そのときは PowerShell も 32bit 用でなければなりません。32bit か 64bit かは **[System.Environment]::Is64BitProcess** で確認できます。

3　実行権限

標準搭載ですが PC によってはスクリプトが実行できないよう権限が設定されていることがあります。スクリプトは DOS のバッチ・ファイルのようにコマンドの記述されたテキスト・ファイルです。拡張子は DOS では**.bat** でしたが PowerShell では**.ps1** です。現在の実行権限は **Get-ExecutionPolicy** で確認し、**Set-ExecutionPolicy** で設定します。実行権限にはいくつか種類があり私の PC では **RemoteSigned** で、スクリプトの実行ができます。ただしインターネットや共有サーバーからダウンロードしたスクリプトにはデジタル署名が必要となります。

3.1　例

現在のユーザーに対して権限を設定し、結果を確認する。（全ユーザに対して設定するには管理者権限が必要）

```
PS> Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser


PS>
PS>
PS> Get-ExecutionPolicy
RemoteSigned
```

4　DOS コマンドとの比較

DOS でできることはすべて PoserShell でできる。DOS に比べて PoserShell のコマンドは長いがタブ補間が効くので見た目ほどタイピングは大変ではない。

DOS	PowerShell
DIR	Get-ChildItem
DEL	Remove-Item
CD	Set-Location
ECHO	Write-Output
DATE	Get-Date
	Set-Date
FOR	foreach
	ForEach-Object

4.1　タブ補間

コマンドの途中で TAB キーを押すと候補が表示される。頭２、３文字打てばたいていのコマンドは入力することができる。これはコマンド本体だけでなくオプションにもその設定値に対しても作動する。

4.2　エイリアスの使用

DOS に慣れたユーザーのために、またキー入力を少なくするために、別名 (エイリアス）を持つコマンドがある。dir や cd くらいなら問題ないが、**%**(For-EachObject) や**?**(Where-Object) となるとある程度 PowerShell を使い慣れた人でないと想像がつかず特殊文字であるために検索も難しい。他人と未来の自分のためにエイリアスは使わないことを勧める。一番時間かかるのは PC の性能不足や最適化されていないプログラムコードではなく人間が悩んで考えている時間であるから。同様の理由でオプションの省略も好ましくない。

5　プロンプト

プロンプトとはコマンドラインの左端に常に表示される文字列です。デフォルトではこれがカレント（現在）ディレクトリのフルパスになっており、深いフォルダで作業するときはかなり邪魔ですのでこれを短くする方法を記します。PowerShell のプロンプト文字列は、prompt 関数によって定義されていますのでそれの上書きによって変更することができます。

```
PS C:\Users\E.WAKABAYASHI-AA> function prompt{ "PS>"}

PS>
```

プロンプトが「PS C:\Users\E.WAKABAYASHI-AA>」から「PS>」に変わりました。しかしこのままでは PowerShell を立ち上げ直すとまたもとに戻ります。恒久的に変えるには PowerShell 起動時に実行される起動スクリプトに prompt 関数を書く必要があります。起動スクリプトのフルパスは**$profile** で取得できます。

```
PS>$profile
N:\Documents\E.WAKABAYASHI-AA\Documents\WindowsPowerShell\Microsoft.PowerShellISE_profile.ps1
```

5.1　カレントディレクトリの確認

pwd(Get-Location コマンドの別名）で確認できます。

```
PS> pwd

Path
----
C:\Users\E.WAKABAYASHI-AA\Desktop\新しいフォルダー
```

6 ヘルプ

6.1 コマンドを探す

「file」を含むコマンドが表示される。

```
PS> Get-Command *file*

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           Write-FileSystemCache                              2.0.0.0    Storage
Function        Block-FileShareAccess                              2.0.0.0    Storage
Function        Clear-FileStorageTier                              2.0.0.0    Storage
Function        Close-SmbOpenFile
.
.
```

6.2 ヘルプを表示する

「Get-ChildItem」のヘルプがブラウザ (IE) で表示される。

```
Get-Help -Online Get-ChildItem
```

7 PowerShell スクリプト

DOS のバッチ・ファイルに相当する。拡張子は**.ps1**。権限設定によっては実行ができない。

7.1 作業の流れ

1. 実行権限を **RemoteSigned** に設定する。
2. 拡張子**.ps1** を **Windows PowerShell ISE** に関連づける。
3. 新しいテキスト。ファイルを作成し、拡張子を**.ps1** に変更する。
4. ダブル・クリックすると **Windows PowerShell ISE** が開く。

7.2 デバッガー

Windows PowerShell ISE には簡単なデバッガーが含まれている。

ショートカットキー	メニューと動作内容
Ctr + 1	スクリプト ウィンドウを上に表示
Ctr + 2	スクリプト ウィンドウを右側に表示
Ctr + 3	スクリプト ウィンドウを最大表示
Ctr + I	スクリプト ウィンドウに移動
Ctr + D	コンソールに移動
F5	実行
Shift + F5	デバッガーを中止
F9	ブレークポイントの設定/解除
F10	ステップオーバー

8 スクリプトの実行

スクリプト aaa.ps1 を実行する方法。

8.1 Windows PowerShell ISE から

拡張子の関連付けがされていれば aaa.ps1 をダブルクリックで ISE が開くのでそのまま F5 キーを押す。

8.2 Windows PowerShell から

ファイル名 aaa.ps1 をコマンドラインにタイプしエンターキーを押す。

```
PS> aaa.ps1
```

8.3 コマンドプロンプト (DOS 窓) から

powershell.exe -File aaa.ps1 とタイプしエンター・キーを押す。

```
> powershell.exe -File aaa.ps1
```

8.4 コマンドプロンプト (DOS 窓) のショートカットから

ショートカットのプロパティを開き以下の設定を行う。

- リンク先: `C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -NoExit -File aaa.ps1`
- 作業フォルダー: (空白)

powershell.exe をプルパスで指定しているが実際には powershell.exe とだけ入力して OK を押せば自動的にフルパスとなる。

「-NoExit」は実行後もウィンドウを開いたままにしておくオプションである。その他の powershell.exe のオプションはコマンドプロンプトで「powershell.exe /?」をタイプすると表示される。

9 パイプとオブジェクト

PowerShell はパイプでオブジェクトを流すことができます。この言葉の意味がわからなければ PowerShell の真価は分かりません。

9.1 パイプ

LINUX のような UNIX 系 OS では、一つの巨大な多機能のプログラムではなく、複数の単機能の小さなプログラムを組み合わせてデータを処理するのが標準的な手法です。プログラムとプログラムはパイプでつなぎます。先のプログラムの出

力がそのまま次のプログラムの入力となるのです。

データの受け渡しはパイプで行われるので中間ファイルを必要としません。文字通りデータがパイプの中を流れるようにプログラムからプログラムへ流れていきます。

UNIX のパイプを流れるのはテキスト（文字）データです。PwerShell のパイプを流れるのはオブジェクトです。より複雑なデータを処理することができます。

9.2 オブジェクト

他で多く語られていますのでその定義には触れません。具体的に PowerShell におけるオブジェクトとは**.NET Framework** です。ざっくりした表現をすればそれは開発者にとっての Windows そのものです。一般ユーザーがマウスとキーボードで行う操作をすべてプログラムとして実行することができます。それ以外に PowerShell 独自の **powershellcustomobject** があります。構造化したテキストデータをパイプに流すことにより柔軟なデータ処理を行うことができます。

10 関数 (Function)

About Functions
https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_functions?view=powershell-6

関数は命令文のリストに名前を付けたもの。コマンド・プロンプトから実行できる。

> A function is a list of PowerShell statements that has a name that you assign. When you run a function, you type the function name. The statements in the list run as if you had typed them at the command prompt.

11 Begin, Process, End

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_functions?view=powershell-6#piping-objects-to-functions

関数はパイプから入力をとることができ、Begin, Process, End のキーワードで動作を制御できる。

> Piping Objects to Functions Any function can take input from the pipeline. You can control how a function processes input from the pipeline using Begin, Process, and End keywords. The following sample syntax shows the three keywords:
>
> ```
> function <name> {
>   begin {<statement list>}
>   process {<statement list>}
>   end {<statement list>}
> }
> ```

Begin は最初に 1 回、End は最後に 1 回、Process は毎回実行される。

> The Process statement list runs one time for each object in the pipeline. While the Process block is running, each pipeline object is assigned to the **$_** automatic variable, one pipeline object at a time.

$_ にはパイプライン・オブジェクトが割り当てられる。

12 オペレーター

About Operators https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_operators?view=powershell-6

+, -, *, /, % のような算術オペレータ以外の特殊なオペレータについて記す。

12.0.1 Call operator &

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_operators?view=powershell-6#call-operator-

> Runs a command, script, or script block. The call operator, also known as the "invocation operator," lets you run commands that are stored in variables and represented by strings or script blocks. The call operator executes in a child scope.

```
PS> $c = "get-executionpolicy"
PS> $c
get-executionpolicy
PS> & $c
AllSigned
```

コマンド、スクリプト、スクリプト・ブロックを実行する。

12.0.2 Comma operator ,

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_operators?view=powershell-6#comma-operator-

> As a binary operator, the comma creates an array. As a unary operator, the comma creates an array with one member. Place the comma before the member.

```
$myArray = 1,2,3
$SingleArray = ,1
```

「,」(カンマ) は配列を作成する。例えば「1,2,3」。「,4」もまた大きさが 1 の配列である。カンマの使用で注意すべき点は関数の引数である。PowerShell の関数の引数は通常のプログラミング言語のように引数間をカンマでくぎらない。関数というよりはコマンドのように複数の引数は空白で区切る。もし PowerShell の関数の引数をカンマで区切った場合、それは複数の引数ではなく、一つの配列であると解釈される。「command aaa, bbb, ccc」 は引数が 3 個あるのではなく 1 個とみなされる。

13 コマンドの使用例

13.1 指定したフォルダ内を再帰的に調べる

子ファイル/ディレクトリを取得する Get-ChildItem コマンドに **-Recurse** オプションを付けると取得範囲がサブ・ディレクトリにまで広がります。PowerShell を使う理由の半分はこの**-Recurse** オプションにあるといっても過言ではありません。

```
Get-ChildItem -Recurse -LiteralPath <フォルダ>
```

13.2 ディレクトリ情報を取得する。

Get-Item コマンドでディレクトリ情報を取得します。PowerShell が扱うのはすべてオブジェクトですので取得したディレクトリ情報もまたオブジェクトです。オブジェクトのタイプをしるために **GetType** コマンドを使用します。**System.IO.FileSystemInfo** クラスの詳細はマイクロソフトのサイトで詳しく記されています。ここではかなり長いパスを指定していますがパス名にもタブ補間が効くので入力は容易です。

```
PS> Get-Item -LiteralPath \\bsu01119\CureSim\加硫流動シミュレーション\★シミュレーションデータベース\MPD


    ディレクトリ: \\bsu01119\CureSim\加硫流動シミュレーション\★シミュレーションデータベース


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        2019/08/27     13:05                MPD



PS> (Get-Item -LiteralPath \\bsu01119\CureSim\加硫流動シミュレーション\★シミュレーションデータベース\MPD).GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     DirectoryInfo                            System.IO.FileSystemInfo
```

13.3 エクセルファイルの一覧を取得し CSV ファイルに出力する

行末の 「|」 はパイプです。コマンドをパイプでつなげています。取得したディレクトリ情報からその配下のファイル情報を取得し、拡張子「.xlsx」でフィルタリングし、ファイル名 (Name) とフルパス（FullName) と更新日（LastWriteTime）を選び、”filelist.csv” に CSV 形式で出力しています。コマンド・オプションの詳細は「Get-Help -Online <コマンド名 >」で確認できます。

```
Get-Item -LiteralPath \\bsu01119\CureSim\加硫流動シミュレーション\★シミュレーションデータベース
\MPD |
Get-ChildItem -Recurse -Filter *.xlsx |
Select-Object -Property Name, FullName, LastWriteTime |
Export-Csv -NoTypeInformation -Encoding Default -LiteralPath filelist.csv -Force
```

13.4 エクセルファイルの更新日時直近 10 個を画面で確認する

前項と同様ディレクトリ情報からファイル情報を取得し、更新日でソート (**Sort-Object -Property LastWriteTime -Descending**) した後、表示数を絞り (**Select-Object -First 10**) リスト形式 (**Format-List**) で表示します。

```
Get-Item -LiteralPath \\bsu01119\CureSim\加硫流動シミュレーション\★シミュレーションデータベース
\MPD |
Get-ChildItem -Recurse -Filter *.xlsx |
Sort-Object -Property LastWriteTime -Descending |
Select-Object -First 10 |
Select-Object -Property Name, FullName, LastWriteTime |
Format-List
```

13.5　CSV 形式のファイル名を作成する

エクセルファイルを CSV 形式でエクスポートするとしてそのファイル名を作成します。

ファイル情報すべてに対して処理を行うために **ForEach-Object** を使います。{} で囲まれたブロックの中では **$_** がパイプから流れてきたオブジェクト（ファイル情報）を表します。ここで **pscustomobject** を作成し、プロパティ EXCEL にはエクセルファイル名、プロパティ CSV には CSV 形式のファイル名を格納します。エクセルファイル名は **$_.Name** です。CSV ファイル名は **$_.Name** の末尾文字列「xlsx」を「csv」に**-replace** で変換しています。

```
Get-Item -LiteralPath \\bsu01119\CureSim\加硫流動シミュレーション\★シミュレーションデータベース
\MPD |
Get-ChildItem -Recurse -Filter *.xlsx |
Select-Object -First 10 |
ForEach-Object {
    [pscustomobject]@{
        EXCEL = $_.Name
        CSV   = $_.Name -replace "xlsx$", "csv"
    }
}
```

14　EXCEL を操作する

.NET には COM 相互運用 (COM Interop) という仕組みがあります。

> .NET には「COM 相互運用」と呼ばれる機能があり、COM コンポーネントを手軽に呼び出すことができる。一方、Excel をはじめとする Office 製品は、その機能をマクロ（VBA）などからも活用できるように COM コンポーネントとして実装されている。このため、COM 相互運用を使えば.NET アプリケーションから容易に Excel や Word のファイルを開き、それをさまざまに操作することが可能だ。
>
> Excel ファイルにアクセスするには？［C#、VB］
> https://www.atmarkit.co.jp/ait/articles/0803/06/news147.html

この仕組みを利用して PowerShell から EXCEL を操作することができます。

14.1　エクセルファイルを開く閉じる

エクセルファイルを開いて閉じるだけのスクリプトです。

```
$filepath = Get-Item -LiteralPath .\bbb.xlsx|Convert-Path
[System.Reflection.Assembly]::LoadWithPartialName("Microsoft.Office.Interop.Excel")|Out-Null
$app = New-Object -ComObject "Excel.Application"
$app.Visible = $true
$book = $app.Workbooks.Open($filepath, $false, $true)
$book.Close($false)
$app.Quit()
```

14.1.1　解説

```
$filepath = Get-Item -LiteralPath .\bbb.xlsx | Convert-Path
```

ファイル名は「bbb.xlsx」ですがフルパスで指定するので **Convert-Path** で変換します。**Get-Item** で取得したファイル情報をパイプで流しています。結果 (フルパス名) は変数 `$filepath` に入ります。

```
[System.Reflection.Assembly]::LoadWithPartialName("Microsoft.Office.Interop.Excel") | Out-Null
```

相互運用のための準備です。**Out-Null** でそのとき表示されるメッセージを捨てています。

```
$app = New-Object -ComObject "Excel.Application"
```

エクセル・アプリケーション・オブジェクトを取得し変数 `$app` に格納します。
Application object
https://docs.microsoft.com/en-us/office/vba/api/excel.application(object)

```
$app.Visible = $true
```

Visible プロパティを真にして表示モードにしています。

```
$book = $app.Workbooks.Open($filepath, $false, $true)
```

エクセル・ファイルを読み取り専用で開き、得られた Workbook オブジェクトを変数 `$book` に格納します。
Workbooks.Open method
https://docs.microsoft.com/en-us/office/vba/api/excel.workbooks.open

```
$book.Close($false)
```

Workbook オブジェクト (エクセルファイル) を閉じます。

```
$app.Quit()
```

エクセルを終了します。

14.2 プロセスの確認と終了

エクセルが起動するとそのプロセスも起動しエクセルの終了と同時にプロセスも終了します。しかし PowerShell からエクセルを操作するスクリプトを開発していると、デバッガーによる強制終了などが原因でエクセルは終了してもそのプロセスが終了せずにプロセスだけが残ることが度々あります。そうなるとダブルクリックでエクセルファイルを開いたときそれ以外のファイルも同時に開くという現象が起こります。そのときはプロセスを終了させる必要があります。

14.2.1 プロセスの確認

プロセスは **Get-Process** で確認します。エクセルが起動しているときはそのプロセスが表示されます。もしエクセルが起動していないときにプロセスが表示されればそれは終了しなければなりません。

```
PS> Get-Process -Name EXCEL

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
    794      50    43560      60416       2.94   11964   1 EXCEL
```

14.2.2 プロセスの終了

```
PS> Stop-Process -Name EXCEL
```

14.3 複数ファイルの処理

単一ファイルの処理ならエクセル VBA を使えばよい。PowerShell は複数ファイルの一括処理ができる。指定したフォルダ内、-Recurse オプションを使えばサブディレクトリまで含めて一括処理を行うことができる。フォルダ内のすべてのエクセルファイルのファイル名と 1 枚目のシート名とその A1 セルの内容を表示するスクリプトを記す。

```
Get-ChildItem -LiteralPath .\excelfiles -Filter *.xlsx |
& {
    begin {
        [System.Reflection.Assembly]::LoadWithPartialName("Microsoft.Office.Interop.Excel")|Out-Null
        $app = New-Object -ComObject "Excel.Application"
    }
    process {
        $filepath = $_.FullName|Convert-Path
        $book = $app.Workbooks.Open($filepath, $false, $true)
        $sheet = $book.Worksheets.Item(1)
        [pscustomobject]@{
            FILE  = $_.Name
            SHEET = $sheet.Name
            A1    = $sheet.Range("A1").Text
        }|Write-Output
        $book.Close($false)
    }
    end {
        $app.Quit()
    }
}
```

Get-ChildItem で取得したエクセルファイル情報をパイプでスクリプトブロックに流す。スクリプト・ブロックにはコール・オペレータ「&」を付けて実行可能にしている。begin ブロックでエクセル・オブジェクトを取得し、process ブロックで各ファイルに対する処理を行い、最後に end ブロックでエクセル・オブジェクトを終了する。

15 使用例 (スクリプト)

15.1 ファイルの一覧を取得する

```
$modelcsv = Get-Item -LiteralPath work|Convert-Path|Join-Path -ChildPath "model.csv"

$d1 = Get-Item -LiteralPath "\\bsu01119\CureSim\加硫流動シミュレーション\★シミュレーションデータベース
\MPD" $d2 = Get-Item -LiteralPath "\\bsu01119\CureSim\加硫流動シミュレーション\★シミュレーションデータ
ベース\proto"

$d1, $d2|
ForEach-Object{
```

```
    $_|Get-ChildItem -Recurse -File -Filter d3plot
}|
ForEach-Object {
    "."|Write-Host -NoNewline

    [pscustomobject]@{
        MODEL  = $_.FullName.Split('\')[7] -replace " .+$", ""
        D3PLOT = $_.FullName
        DIR    = $_|Convert-Path|Split-Path -Parent
    }
}|
ForEach-Object {
    foreach ($i in Get-ChildItem -LiteralPath $_.DIR -Filter *.xls?) {
        [pscustomobject]@{
            MODEL  = $_.MODEL
            DIR    = $_.DIR
            D3PLOT = $_.D3PLOT
            EXCEL  = $i.FullName
        }|Write-Output
    }
}|
#Format-List
Export-Csv -Encoding Default -NoTypeInformation -LiteralPath work\model.csv

""|Write-Host
```

15.2 ファイル・リスト内のエクセル・ファイルのシート名を取得する

```
Import-Csv -Encoding Default -LiteralPath .\work\model.csv|
Where-Object {($_.EXCEL|Split-Path -Leaf) -cmatch "_pressure_PLY"}|
&{
    begin {
        [System.Reflection.Assembly]::LoadWithPartialName("Microsoft.Office.Interop.Excel")|Out-Null
        $excel = New-Object -ComObject "Excel.Application"
        $excel.Visible = $false
    }

    process {
        "."|Write-Host -NoNewline
        $o = $_
        try {
            $book  = $excel.Workbooks.Open($o.EXCEL, $false, $true)
            $book.Sheets|
            ForEach-Object {
                [pscustomobject]@{
                    MODEL  = $o.MODEL
```

```
                    SHEET  = $_.Name
                    EXCEL  = $o.EXCEL
                    D3PLOT = $o.D3PLOT
                    DIR    = $o.DIR
                }|Write-Output
            }
            $book.Close($false)
        } catch {
            "Error"
            $o.EXCEL
            exit 1
        }
    }

    end {
        $excel.Quit()
    }
}|
#Format-List
Export-Csv -NoTypeInformation -Encoding Default -LiteralPath "work\sheet_ply.csv"

""|Write-Host
```

15.3 エクセル・シートの指定した領域を CSV ファイルに書き出す

```
$sim = Get-Item -LiteralPath .\work\sim1.xlsx

# セル内改行を取り除く
function remove_crlf_comma($str) {
    $a = ""
    ([string]$str).GetEnumerator().ForEach({
        # 制御コードを除外
        if ([System.Char]::GetUnicodeCategory($_) -ne [System.Globalization.UnicodeCategory]::Control) {
            if ($_ -eq ',') {
                $a += '、'
            } else {
                $a += $_
            }
        }
    })
    return $a
}


[System.Reflection.Assembly]::LoadWithPartialName("Microsoft.Office.Interop.Excel")|Out-Null
$excel = New-Object -ComObject "Excel.Application"
```

```
$excel.Visible = $false

$book  = $excel.Workbooks.Open($sim.FullName, $false, $true)
$excel.ActiveWindow.FreezePanes = $false

$sheet = $book.Sheets("全体管理表")

$model_rg = $sheet.Range("B8")
$jul1_rg  = $sheet.Range("Z8")
$dec31_rg  = $sheet.Range("HF8")
$tanto_rg  = $sheet.Range("T8")
$status_rg = $sheet.Range("U8")

if ($model_rg.Text -ne "モデル No.") {
    "セルが変更された。"
    exit 1
}

if ($jul1_rg.Text -ne "1/1") {
    "セルが変更された。"
    exit 1
}

if ($dec31_rg.Text -ne "7/8") {
    "セルが変更された。"
    exit 1
}

if ($tanto_rg.Text -ne "DR 担当") {
    "セルが変更された。"
    exit 1
}

if ($status_rg.Text -ne "進捗") {
    "セルが変更された。"
    exit 1
}



$header_row = $model_rg.Row
$top_row    = $header_row + 1
$bottom_row = $model_rg.End([Microsoft.Office.Interop.Excel.XlDirection]::xlDown).Row

$model_col = $model_rg.Column

$date_rgs = $sheet.Range($jul1_rg, $dec31_rg)
```

```
$jul1_col  = $jul1_rg.Column
$dec31_col = $dec31_rg.Column


$r1 = $sheet.Cells($header_row, $jul1_col)
$r2 = $sheet.Cells($header_row, $dec31_col)
$DATELINE = $sheet.Range($r1, $r2)

function find($str, $range_of_line) {

    # "0"とかなら数値型に変換する。
    if ($str -match "^\d+$"){
        $str = [int]$str
    }

    try{
        $match     = $excel.WorksheetFunction.Match($str, $range_of_line, 0)
        $match_rg  = $excel.WorksheetFunction.Index($DATELINE, $match)
        $match_day = $excel.WorksheetFunction.Text($match_rg, "yyyy-mm-dd")
    }
    catch{
        $match_day = ""
    }
    return $match_day
}


$top_row .. $bottom_row|
ForEach-Object {
    $r = $_
    $r1 = $sheet.Cells($r, $jul1_col)
    $r2 = $sheet.Cells($r, $dec31_col)
    $rg12 = $sheet.Range($r1, $r2)

        $model    = $sheet.Cells($r, $model_col).Text
        $size     = $sheet.Cells($r, $model_col + 1).Text
        $seisan   = $sheet.Cells($r, $model_col + 4).Text
        $process  = $sheet.Cells($r, $model_col + 5).Text
        $factory  = $sheet.Cells($r, $model_col + 6).Text
        $mokuteki = $sheet.Cells($r, $model_col + 10).Text
        $tanto    = $sheet.Cells($r, $tanto_rg.Column).Text
        $status   = $sheet.Cells($r, $status_rg.Column).Text
        $zeroday  = find "0"  $rg12
        $drday    = find "DR" $rg12

        $model    = remove_crlf_comma $model
```

```
        $size     = remove_crlf_comma $size
        $seisan   = remove_crlf_comma $seisan
        $process  = remove_crlf_comma $process
        $factory  = remove_crlf_comma $factory
        $mokuteki = remove_crlf_comma $mokuteki
        $tanto    = remove_crlf_comma $tanto
        $status   = remove_crlf_comma $status
        $zeroday  = remove_crlf_comma $zeroday
        $drday    = remove_crlf_comma $drday


    [pscustomobject]@{
MODEL    = $model
SIZE     = $size
SEISAN   = $seisan
PROCESS  = $process
FACTORY  = $factory
MOKUTEKI = $mokuteki
TANTO    = $tanto
STATUS   = $status
ZERODAY  = $zeroday
DRDAY    = $drday
    }
}|Export-Csv -Encoding Default -NoTypeInformation -LiteralPath .\work\sim1.csv

$book.Close($false)
$excel.Quit()
```

15.4 ダイアログを表示しパラメータを YAML ファイルに保存する。

```
$version = "BareY2_ver4.9.0"  # 複数.dyn 出力をリスト化
#$version = "BareY2_ver4.8.0"  # ノード並びチェック追加
#$version = "BareY2_ver4.7.0"  # VH 位置（隣ノード）を最終タイムステートで検出
#$version = "BareY2_ver4.6.1"  # ノード並びテーブルから XYZ 座標値削除
#$version = "BareY2_ver4.6.0"  # 環境変数でなく log.yaml を介してパラメータを渡す
#$env:VERSION = "BareY2_ver4.5.1"  # LS-PREPOT, RScript はサーバー上を利用
#$env:VERSION = "BareY2_ver4.4.1"  # debug_disp_node.Rmd 他
#$env:VERSION = "BareY2_ver4.4.0"  # モールド・エレメントの重複に対応
#$env:VERSION = "BareY2_ver4.3.1"  # areavh.R STATE_NO の重複削除忘れ
#$env:VERSION = "BareY2_ver4.3.0"  # work=>results を -Recurse で
#$env:VERSION = "BareY2_ver4.2.0"  # モールド・ノード始点検出方法修正
#$env:VERSION = "BareY2_ver4.1.0"  # '$' を含むサーバ・パスに対応
#$env:VERSION = "BareY2_ver4.0.0"  # レンジ毎の最遠タイヤ・ノード
#$env:VERSION = "BareY2_ver3.4.0"  # VHLX, VHRY 自動設定
#$env:VERSION = "BareY2_ver3.3.0"  # 測定範囲指定変更（VHL=>VHLX, VHR=>VHRY）
#$env:VERSION = "BareY2_ver3.2.0"  # 測定範囲をユーザが指定（VHL, VHR）
```

```
#$env:VERSION = "BareY2_ver3.1.0"  # 最終ステートの面積も表示
#$env:VERSION = "BareY2_ver3.0.0"  # 辺で端ノード抽出
#$env:VERSION = "BareY2_ver2.0.0"  # 面積
#$env:VERSION = "BareY2_ver1.2.1"  # デバッグ用に測定ノード（intfor_nodelist）を表示
#$env:VERSION = "BareY2_ver1.2.0"  # 最終 STATE_NO を指定する。
#$env:VERSION = "BareY2_ver1.1.2"  # area.html を nmake.exe からではなく START.ps1 から開く。
#$env:VERSION = "BareY2_ver1.1.1"  # Makefile の Usage 変更

if ($args[0] -eq "-v") {
    $env:VERSION
    exit
}

$simdir = Get-Item -LiteralPath \\bsu01119\CureSim\加硫流動シミュレーション\★シミュレーションデータベース
#$simdir = Get-Item -LiteralPath "\\Bcf19065\85\N4600\001_共通管理\006_4650\102_登録テーマ別資料\18-4652-PD-P-05_不良予測技術開発\(2) 開発諸元 (G1)\○ Sim ベア間エア予測\構造&成型機種別_予測精査\サイドベア予測プログラム_評価\20190313_MTG\2. 検証サイズ選定"
#$simdir = Get-Item -LiteralPath "\\BCF19064\N4600a‘$\001_共通管理\006_4650\102_登録テーマ別資料\18-4652-PD-P-05_不良予測技術開発\(2) 開発諸元 (G1)\○ Sim ベア間エア予測\構造&成型機種別_予測精査\サイドベア予測プログラム_評価\20190313_MTG\2. 検証サイズ選定"

[void][System.Reflection.Assembly]::LoadWithPartialName("System.Windows.Forms")

$form1      = New-Object -TypeName System.Windows.Forms.Form
$btn_d3plot = New-Object -TypeName System.Windows.Forms.Button
$btn_vhline = New-Object -TypeName System.Windows.Forms.Button
$btn_ok     = New-Object -TypeName System.Windows.Forms.Button
$lbl_d3plot = New-Object -TypeName System.Windows.Forms.Label
$lbl_intfor = New-Object -TypeName System.Windows.Forms.Label
$lbl_vhline = New-Object -TypeName System.Windows.Forms.Label
$lbl_last   = New-Object -TypeName System.Windows.Forms.Label
$lbl_vhl    = New-Object -TypeName System.Windows.Forms.Label
$lbl_vhr    = New-Object -TypeName System.Windows.Forms.Label
$tbox_last  = New-Object -TypeName System.Windows.Forms.TextBox
$tbox_vhl    = New-Object -TypeName System.Windows.Forms.TextBox
$tbox_vhr    = New-Object -TypeName System.Windows.Forms.TextBox
$ofd_d3plot = New-Object -TypeName System.Windows.Forms.OpenFileDialog
$ofd_vhline = New-Object -TypeName System.Windows.Forms.OpenFileDialog

$font = [System.Drawing.Font]::new($lbl_d3plot.Font.FontFamily.Name, 11)
$btn_d3plot.Font = $font
$btn_vhline.Font = $font
$btn_ok.Font     = $font
$lbl_d3plot.Font = $font
$lbl_intfor.Font = $font
$lbl_vhline.Font = $font
```

```
$lbl_last.Font   = $font
$lbl_vhl.Font    = $font
$lbl_vhr.Font    = $font

$form1.Text   = $env:VERSION
$form1.Width  = 850
$form1.Height = 550

$btn_d3plot.Text = "d3plot"
$btn_vhline.Text = "VH_line.cfile"
$btn_ok.Text     = "OK"
$lbl_d3plot.Text = "D3PLOT = "
$lbl_intfor.Text = "INTFOR = "
$lbl_vhline.Text = "VHLINE = "
$lbl_last.Text   = "LAST_STATE_NO(最終タイム・ステート):"
$lbl_vhl.Text    = "VHLX(測定左端 X 値 [mm])[空白なら自動設定]:"
$lbl_vhr.Text    = "VHRY(測定右下端 Y 値 [mm][空白なら自動設定]):"

$btn_d3plot.Top    = 0
$lbl_d3plot.Top    = 40
$lbl_intfor.Top    = 110
$btn_vhline.Top    = 180
$lbl_vhline.Top    = 210
$lbl_last.Top      = 280
$tbox_last.Top     = 280
$lbl_vhl.Top       = 340
$tbox_vhl.Top      = 340
$lbl_vhr.Top       = 380
$tbox_vhr.Top      = 380
$btn_ok.Top        = 430

$tbox_last.Left    = 250
$tbox_vhl.Left    = 350
$tbox_vhr.Left    = 350

$lbl_d3plot.Height = 50
$lbl_intfor.Height = 50
$lbl_vhline.Height = 50

$btn_d3plot.Width  = 120
$btn_vhline.Width  = 120
$btn_ok.Width      = 120

$lbl_d3plot.Width  = 800
$lbl_intfor.Width  = 800
$lbl_vhline.Width  = 800
$lbl_last.Width    = 250
```

```
$lbl_vhl.Width    = 350
$lbl_vhr.Width    = 350


$ofd_d3plot.InitialDirectory = $simdir.FullName
$ofd_d3plot.FileName = "d3plot"

$tbox_last.Text = "1550"


$btn_d3plot_click = {
    if ($ofd_d3plot.ShowDialog() -eq [System.Windows.Forms.DialogResult]::OK) {
        $script:d3plot = $ofd_d3plot.FileName
        if (Test-Path -LiteralPath $script:d3plot) {
            $script:intfor = [System.IO.Path]::GetDirectoryName($script:d3plot) + "\intfor"
            if (Test-Path -LiteralPath $script:intfor) {
                $lbl_d3plot.Text += $script:d3plot
                $lbl_intfor.Text += $script:intfor
            }
        }
    }
}
$btn_d3plot.Add_Click($btn_d3plot_click)

$btn_vhline_click = {
    if ($ofd_vhline.ShowDialog() -eq [System.Windows.Forms.DialogResult]::OK) {
        $script:vhline = $ofd_vhline.FileName
        if (Test-Path -LiteralPath $script:vhline) {
            $lbl_vhline.Text += $script:vhline
        }
    }
}
$btn_vhline.Add_Click($btn_vhline_click)

$btn_ok_click = {
    $form1.Close()
}
$btn_ok.Add_Click($btn_ok_click)


$form1.Controls.Add($btn_d3plot)
$form1.Controls.Add($lbl_d3plot)
$form1.Controls.Add($lbl_intfor)
$form1.Controls.Add($btn_vhline)
$form1.Controls.Add($lbl_vhline)
$form1.Controls.Add($lbl_last)
$form1.Controls.Add($tbox_last)
```

```
$form1.Controls.Add($lbl_vhl)
$form1.Controls.Add($tbox_vhl)
$form1.Controls.Add($lbl_vhr)
$form1.Controls.Add($tbox_vhr)
$form1.Controls.Add($btn_ok)

$form1.ShowDialog() |Out-Null


[pscustomobject]@{
    VERSION       = $version
    DATE          = [string](Get-Date -Format g)
    D3PLOT        = $script:d3plot
    INTFOR        = $script:intfor
    VHLINE        = $script:vhline
    LAST_STATE_NO = $tbox_last.Text
    VHLX          = $tbox_vhl.Text
    VHRY          = $tbox_vhr.Text
}|
ConvertTo-Yaml|
Set-Content -LiteralPath "work\log.yaml"
```

15.5 CSV ファイルを連結する

```
$sim1 = Get-Content -Encoding Default -LiteralPath .\work\sim1.csv

$sim7 = Get-Content -Encoding Default -LiteralPath .\work\sim7.csv|Select-Object -Skip 1

$sim1 + $sim7|Set-Content -Encoding Default -LiteralPath work/sim.csv
```