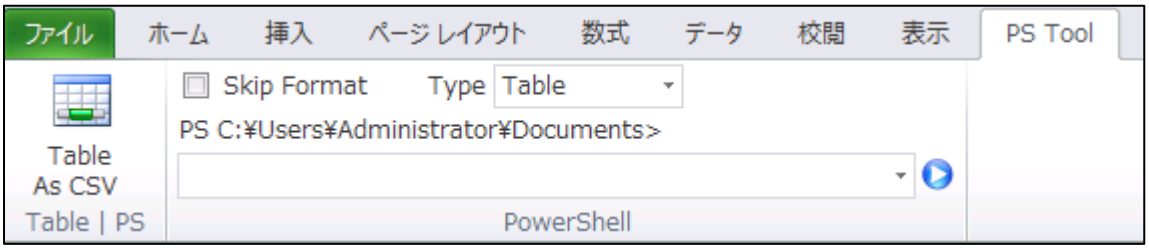
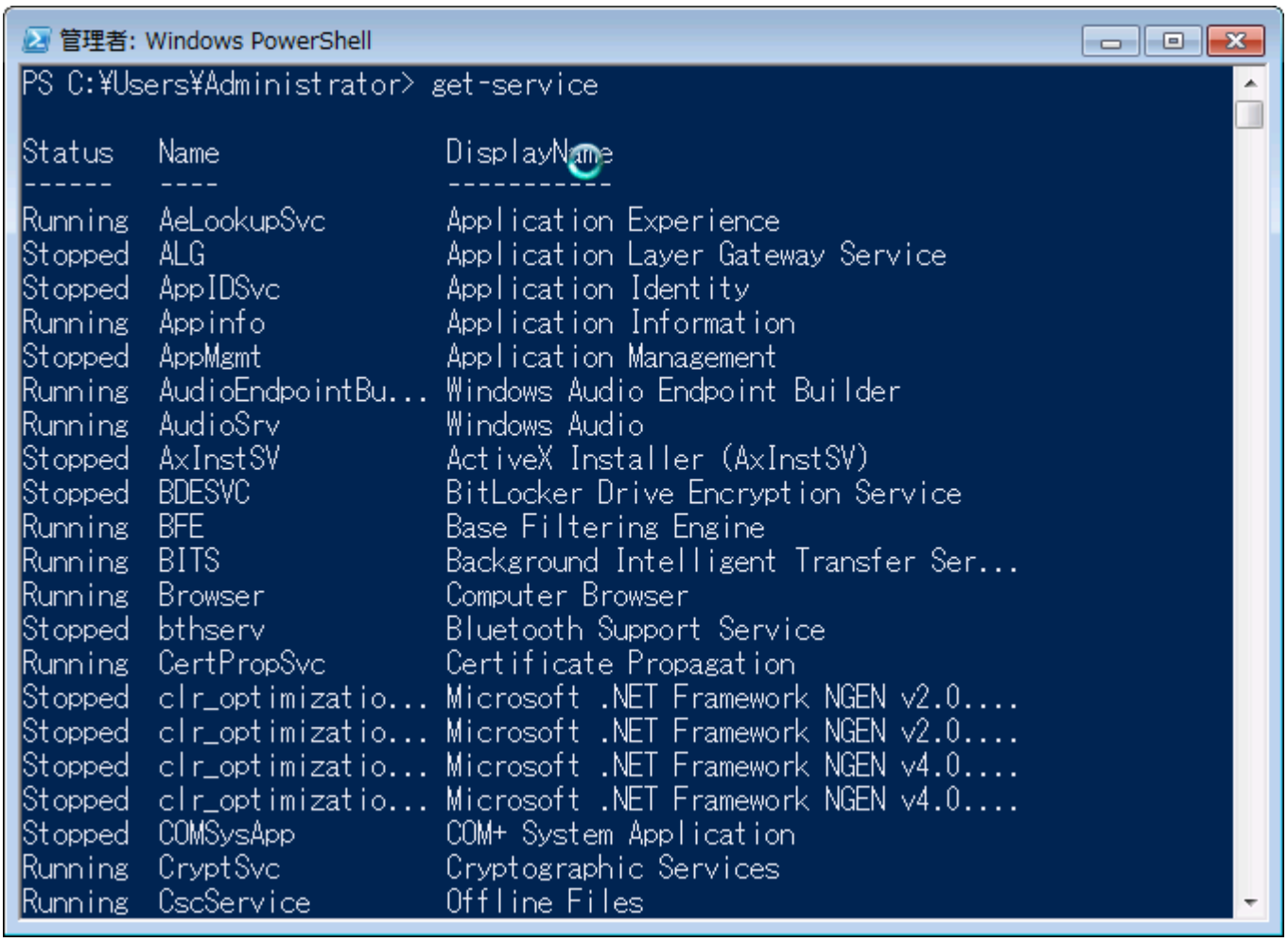
Excel PowerShell Tool

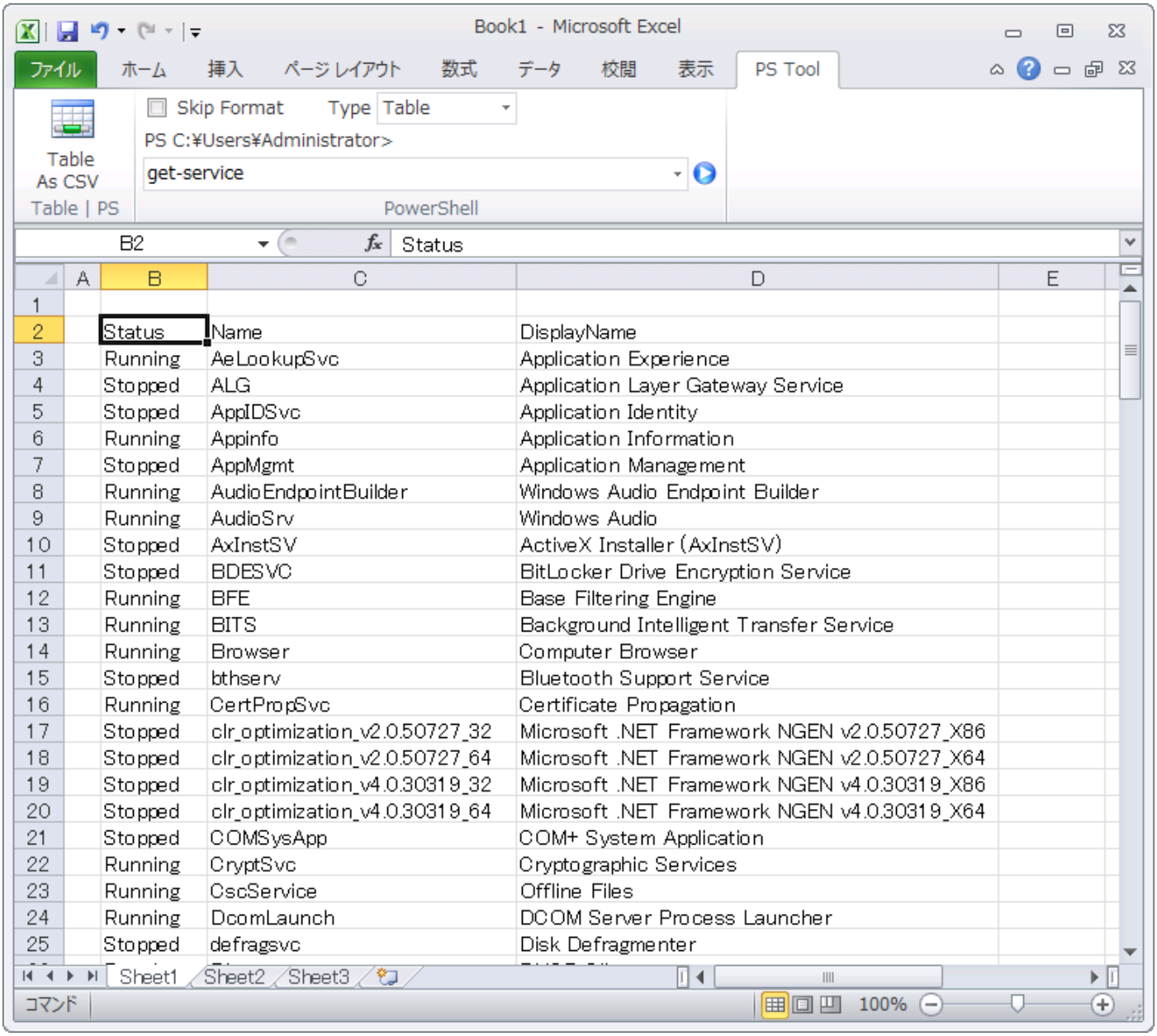
Excel PowerShell Tool(以下 PS Tool)は、Excel から直接 PowerShell コマンドを呼び出すことが出来るアドインです。PS Tool は PowerShell を利用して、データの取得・編集・更新という一連の運用を Excel 上で可能にします。



PS Tool は、PowerShell の出力結果を Excel のセル上に自動的に展開します。例として、PowerShell コンソール上で Get-Service コマンドレットを実行してみます。



同じ事を PS Tool 上で行います。



	A	B	C	D	E
1					
2		Status	Name	DisplayName	
3		Running	AeLookupSvc	Application Experience	
4		Stopped	ALG	Application Layer Gateway Service	
5		Stopped	AppIDSvc	Application Identity	
6		Running	Appinfo	Application Information	
7		Stopped	AppMgmt	Application Management	
8		Running	AudioEndpointBuilder	Windows Audio Endpoint Builder	
9		Running	AudioSrv	Windows Audio	
10		Stopped	AxInstSV	ActiveX Installer (AxInstSV)	
11		Stopped	BDESVC	BitLocker Drive Encryption Service	
12		Running	BFE	Base Filtering Engine	
13		Running	BITS	Background Intelligent Transfer Service	
14		Running	Browser	Computer Browser	
15		Stopped	bthserv	Bluetooth Support Service	
16		Running	CertPropSvc	Certificate Propagation	
17		Stopped	clr_optimization_v2.0.50727_32	Microsoft .NET Framework NGEN v2.0.50727_X86	
18		Stopped	clr_optimization_v2.0.50727_64	Microsoft .NET Framework NGEN v2.0.50727_X64	
19		Stopped	clr_optimization_v4.0.30319_32	Microsoft .NET Framework NGEN v4.0.30319_X86	
20		Stopped	clr_optimization_v4.0.30319_64	Microsoft .NET Framework NGEN v4.0.30319_X64	
21		Stopped	COMSysApp	COM+ System Application	
22		Running	CryptSvc	Cryptographic Services	
23		Running	CscService	Offline Files	
24		Running	DcomLaunch	DCOM Server Process Launcher	
25		Stopped	defragsvc	Disk Defragmenter	

PS Tool がワークシートのフォーカス・セルを起点にして、PowerShell コマンドの出力を自動的に展開していることが分かります。

従来 PowerShell の出力結果を Excel に読み込ませたい場合には、Export-Csv コマンドレットを使う必要がありましたが、文字コードや型の問題に悩まされていました。PS Tool では Excel を介在させることで、変換に伴う多くの問題を解決します。

また、任意のテーブルを CSV ファイルのように扱い、パイプの形で PowerShell に渡す事も可能です。これにより Import-Csv コマンドレットを使う必要がなくなります。

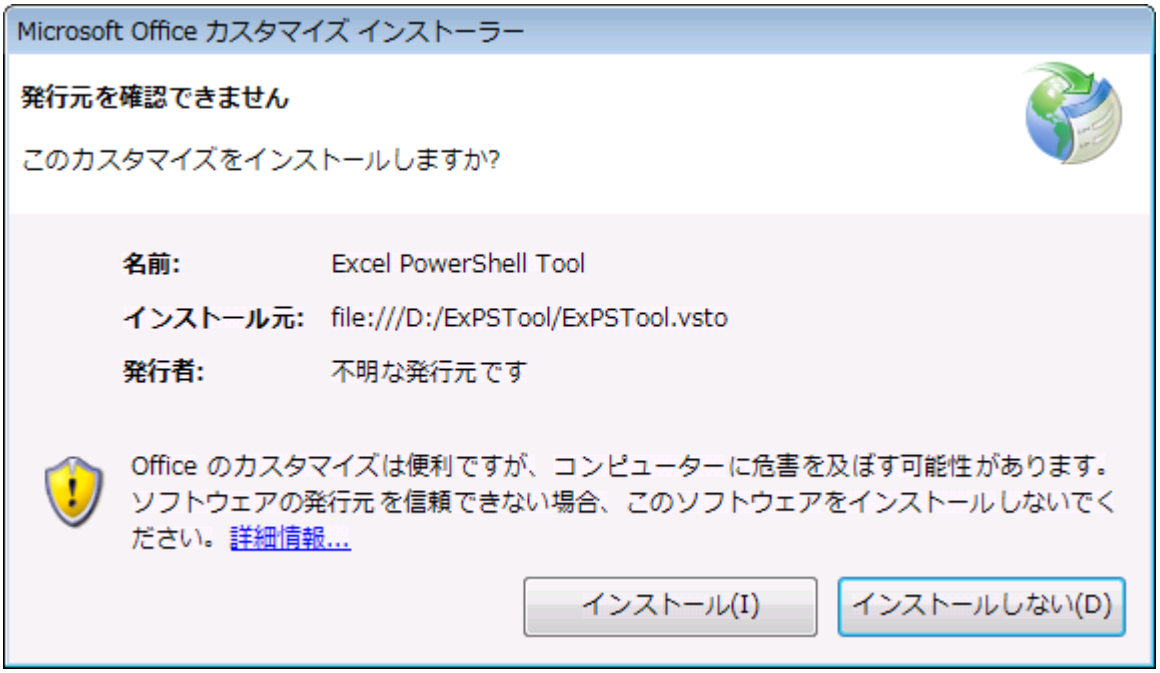
インストール

PS Tool のシステム要件を示します。

OS：Windows 7/8, Server 2008 R2/2012
Excel 2010/2013

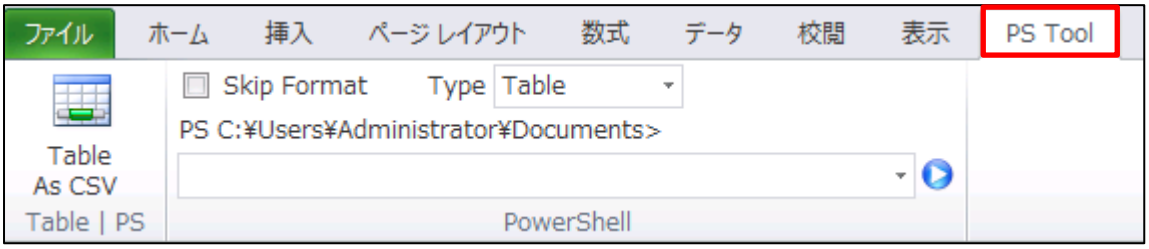
PS Tool は VSTO(Visual Studio Tool for Office)によって作成されていて、そのインストーラーは Click Once で作成されています。以下の手順でインストールを行います。

ダウンロードした ExPSTool.zip を適当なフォルダー(※ここでは D:¥ExPSTool)で解凍し、setup.exe を実行します。

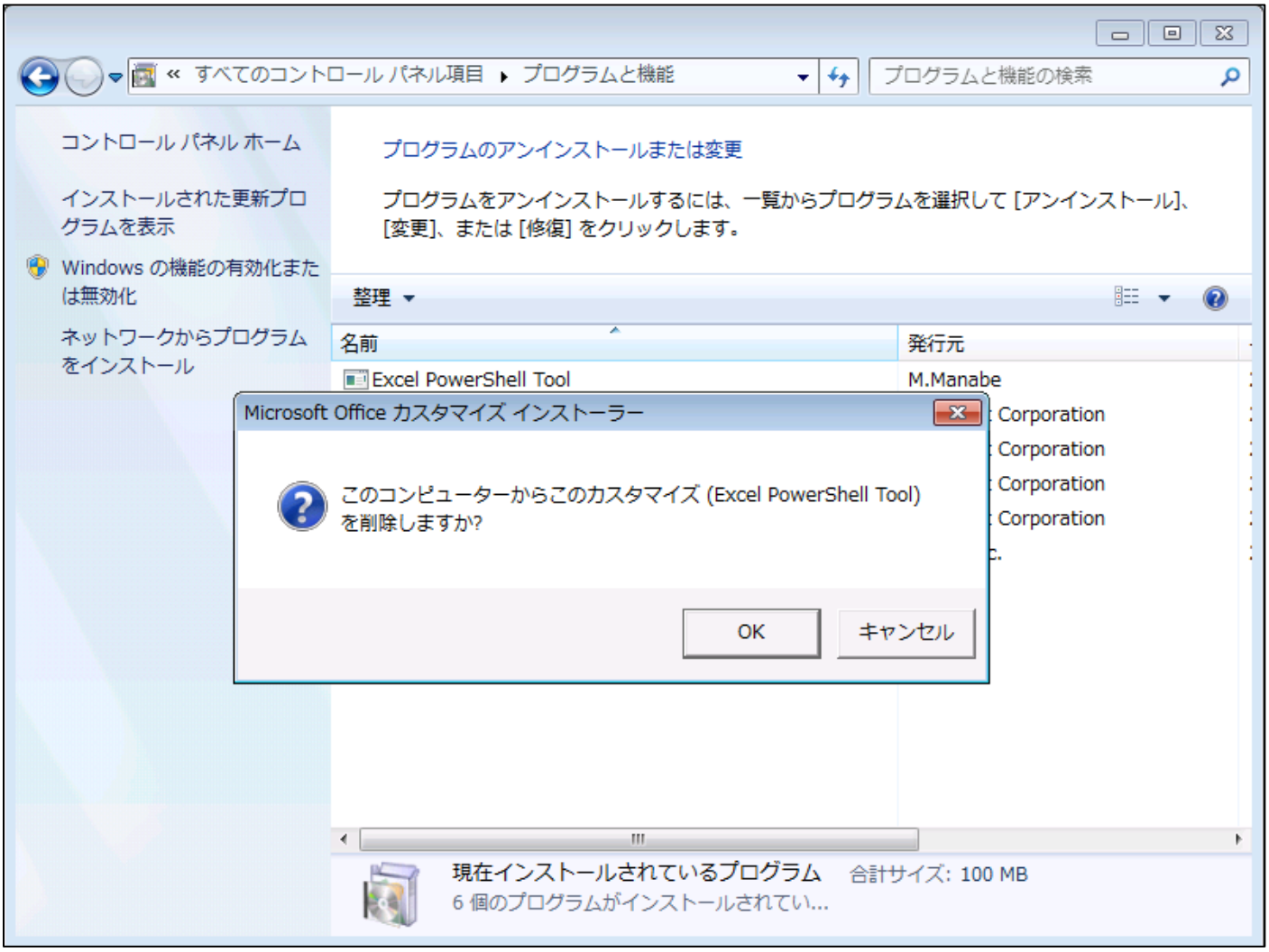


PS Tool では Visual Studio の簡易証明書を利用しているので、上のような警告が出ます。警告の出ない証明書はそれなりの費用がかかります…ここでは警告を無視してください。インストールが終了すると以下の表示がされます。

Excel を起動し、リボンに専用タブ “PS Tool” が有ることを確認します。



アンインストールは”コントロールパネル”-“プログラムと機能”から Excel PowerShell Tool を選んでアンインストールしてください。



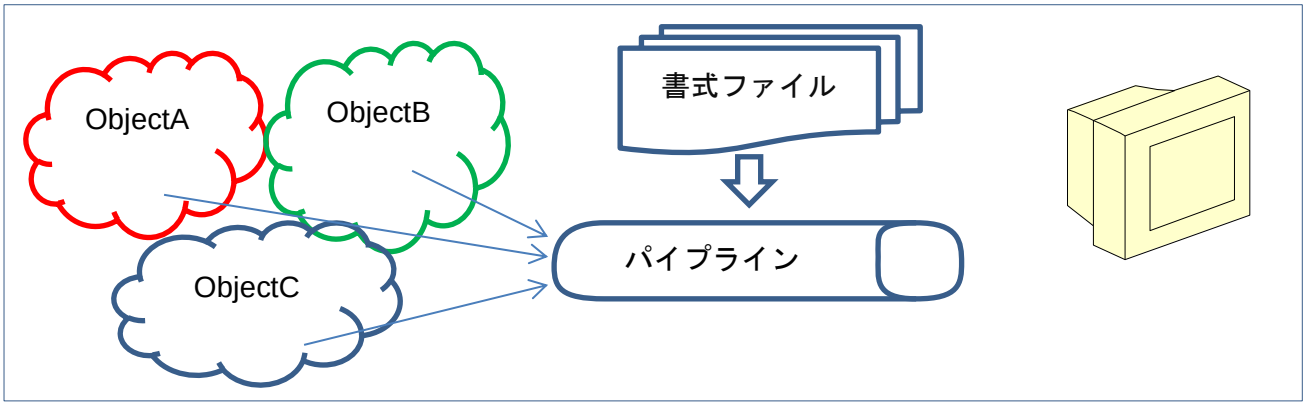
VSTO Click Once の制約で、インストールは個人レベルになります。

PowerShell

PowerShell は.Net Framework 上で動作するシェルで、オブジェクト(の配列)がパイプラインを流れるという特徴があります。オブジェクトの各機能を利用する事で、unix シェルで行うような字句解析が不要となり、効率の良い開発が可能になります。ここでは PS Tool を使用する際に知っておいて欲しい項目のみ説明します。

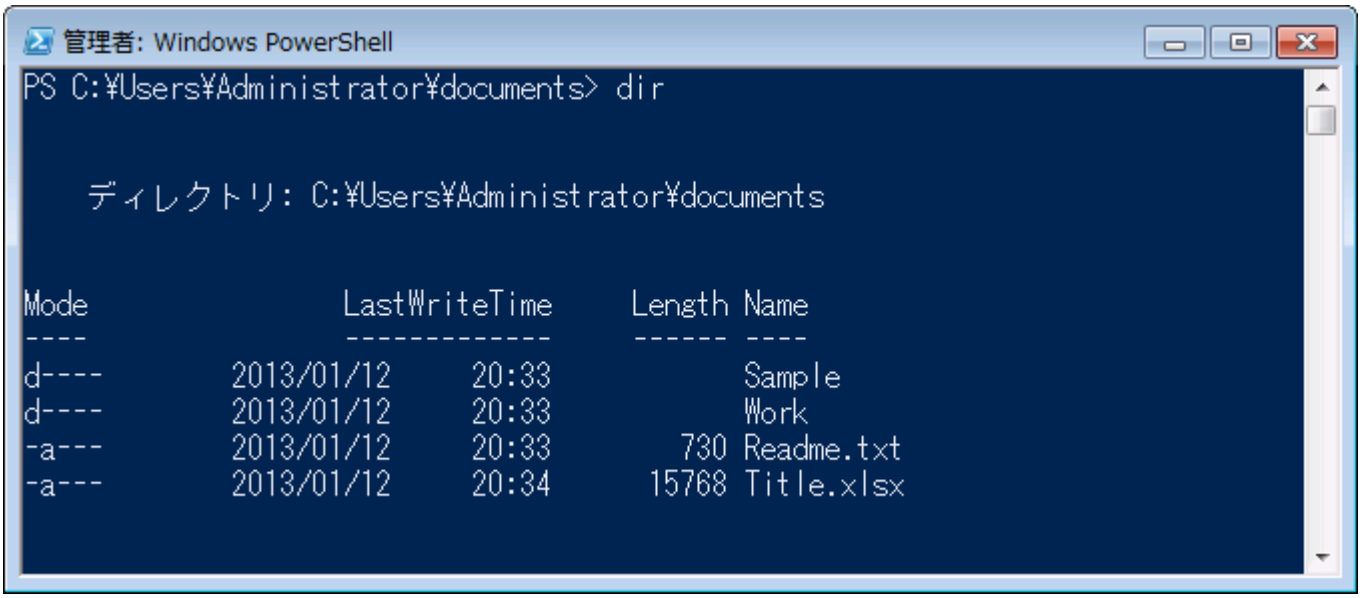
書式

PowerShell ではオブジェクトがパイプラインを流れていますが、コマンドレットの出力としてコンソールに表示されるのは「書式ファイル」によって加工された文字列となります。書式ファイルは C:¥Windows¥System32¥WindowsPowerShell¥v1.0 下に *.format.ps1xml の名前で配置されていて、オブジェクトの型により PowerShell が最適な出力を調整します。



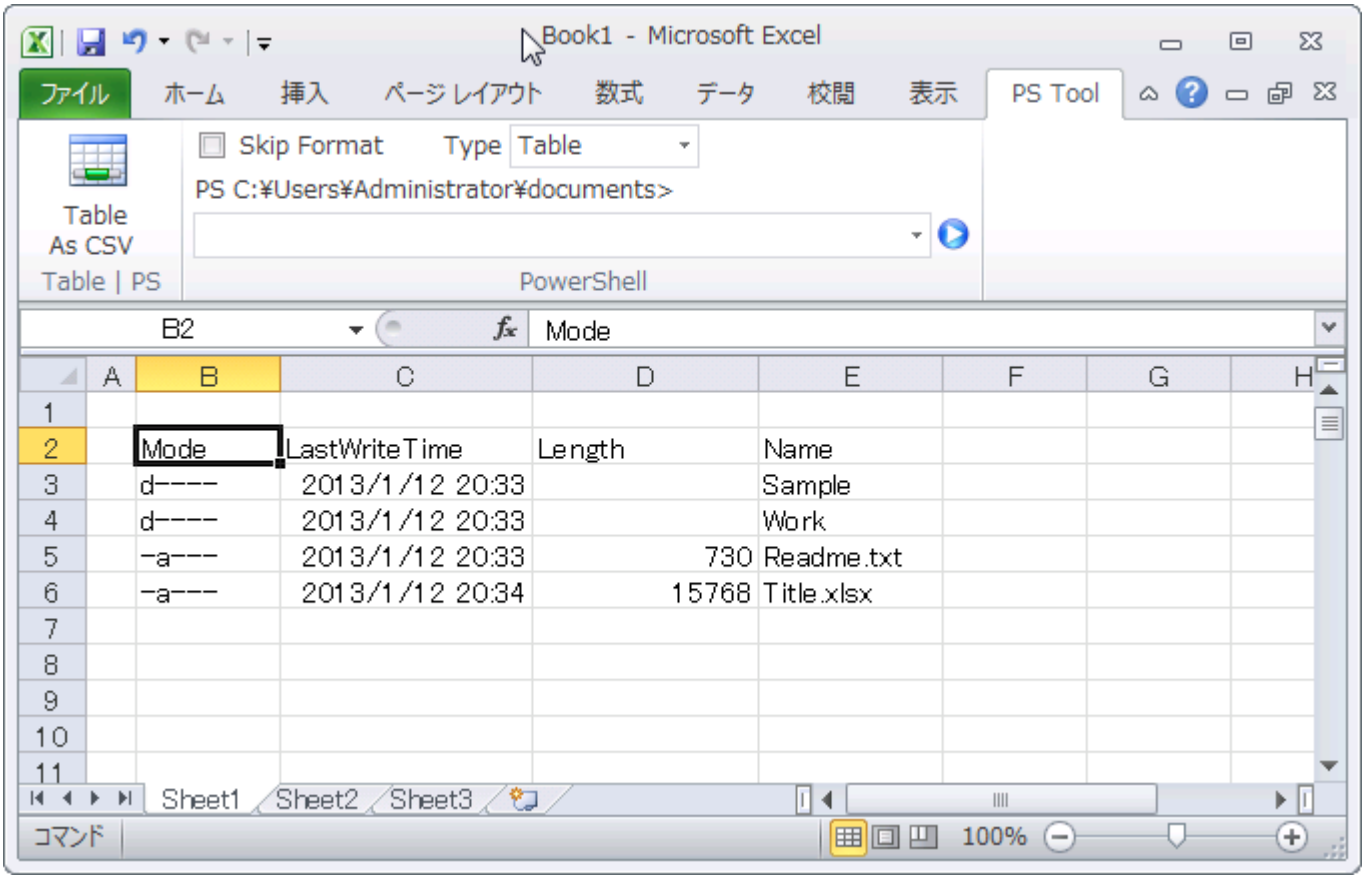
PowerShell のコンソールに表示される文字列は、書式ファイルにより調整された結果であり、全情報ではないと認識することが重要です。

Get-ChildItem(=dir)コマンドレットを例にしてみます。



DOS プロンプト時代と変わらない感じで文字が表示されていますが、実際には “Sample”, “Work”という 2 つの System.IO.DirectoryInfo 型オブジェクト、”Readme.txt”, “Title.xlsx” という 2 つの System.IO.FileInfo 型オブジェクトがパイプを通り、書式ファイルにより “Mode”, “LastWriteTime”, “Length”, “Name” の列名を与えられ、それに対応するプロパティ値(※列名と同じとは限らない)を出力しています。標準の出力書式に問題が有るときは、独自に書式を設定してやれば任意の結果を得ることも可能です。

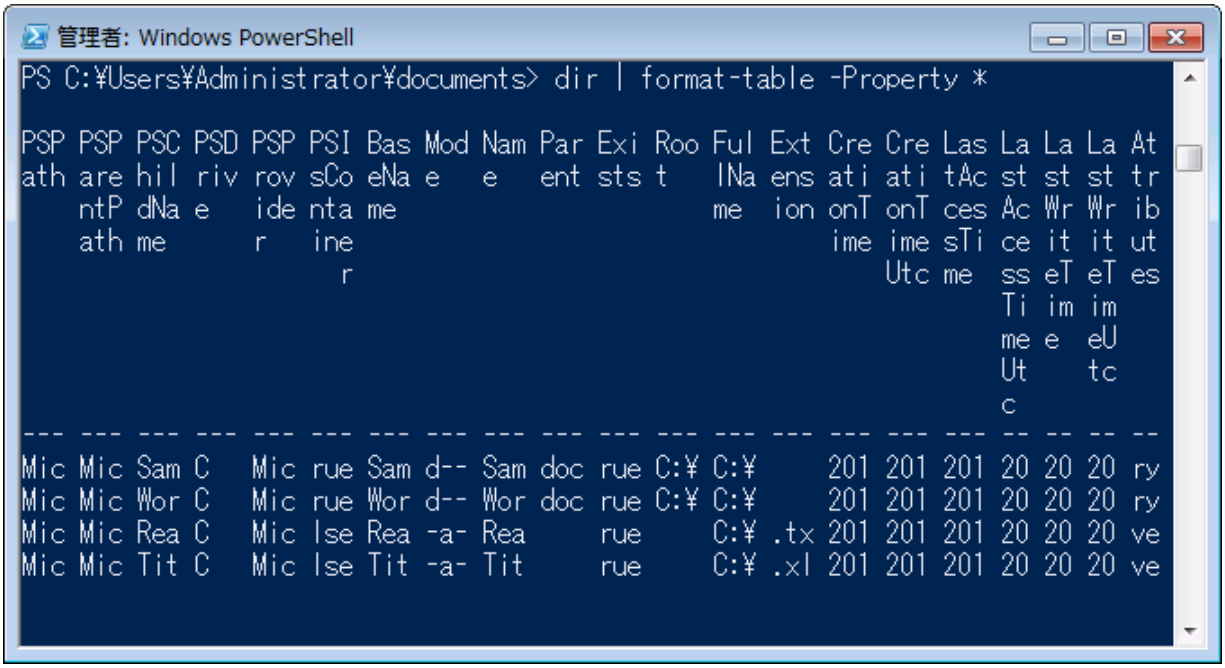
PS Tool では書式ファイルを独自に解析し、(ある程度)PowerShell コンソールと同様の出力をサポートしています。



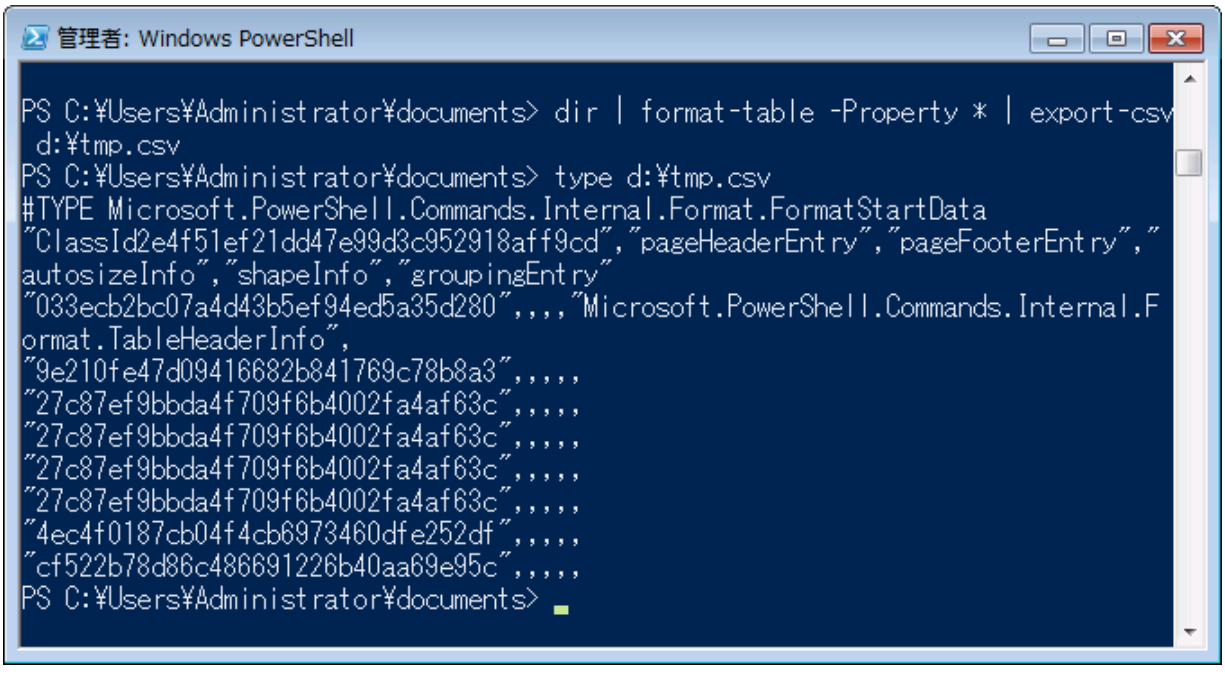
Format-Table/Format-List

Format-Table は表形式、Format-List はリスト形式でオブジェクトのプロパティを抽出・表示します。抽出の点では Select-Object と似ていますが、コンソールサイズに合わせた出力をしてくれるのでユーザー的にはよく使うコマンドレットです。

前述の Get-ChildItem(=dir)コマンドレットですが、Format-Table を使えば任意のプロパティを表示することができます。



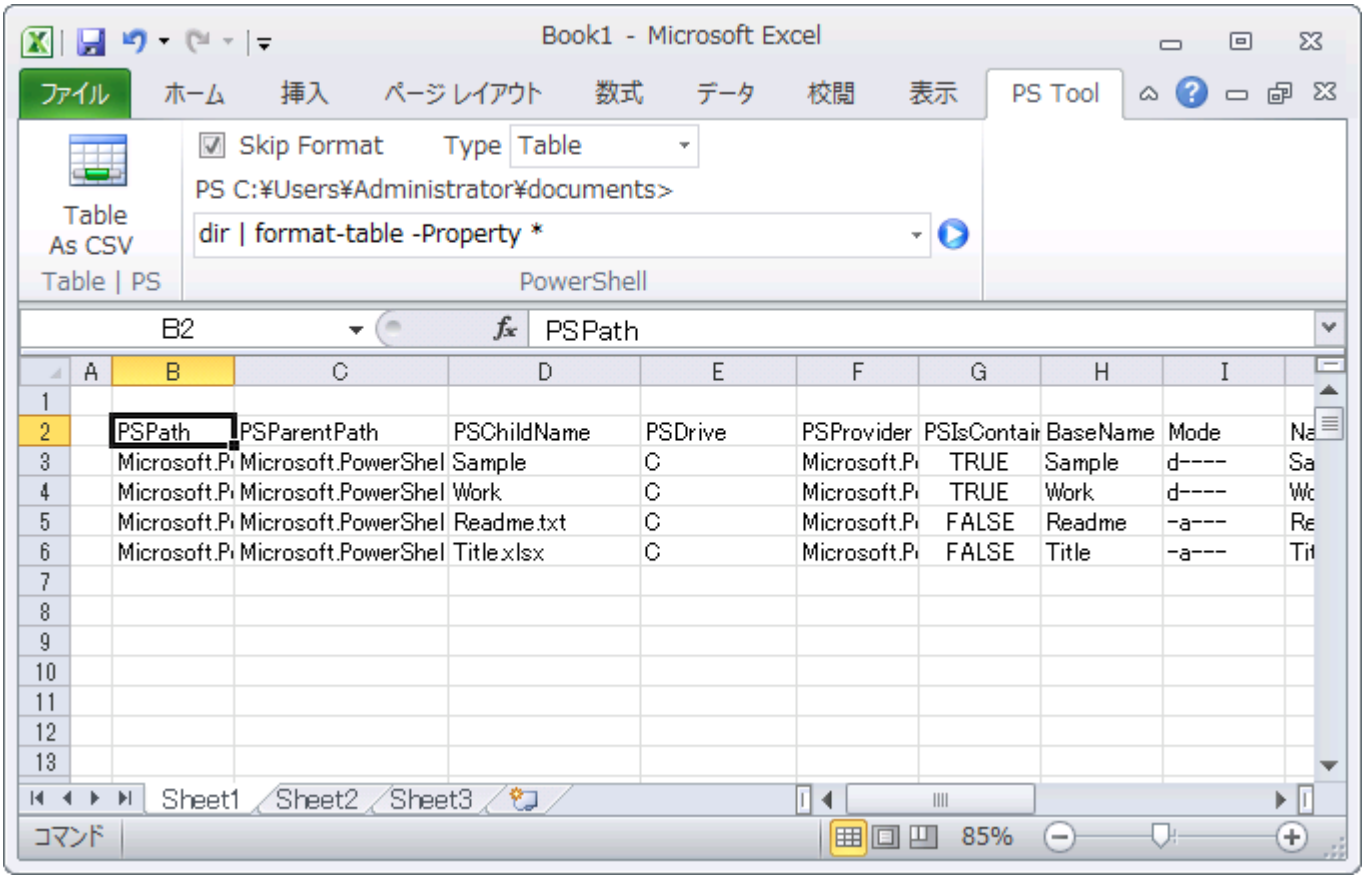
この例ではコンソールサイズが小さすぎて、殆どの表示が省略されているように見えます。しかし、Format-Table の内部的には全ての文字列が保持されていて、出力対象によってどこまで出力するかを調節しています。残念なことに、オブジェクトの内容は独自形式になっているので、パイプで繋いでも希望する結果になりません。例えば、Export-Csv に繋いでみると…



ユーザーとしては、コンソールに現れる形で CSV ファイルが生成されることを期待しますが、意味不明な文字列が出てくるだけです。この場合 Select-Object を使う事で希望する CSV ファイルが得られますが、Select-Object の出力はリスト形式であり、感覚的に使いにくいです。

PowerShell では、コンソールへの出力としての文字列より、その裏に隠れているオブジェクトをイメージすることが重要ですが、unix シェルでは画面に表示される文字列が真なので、この発想の違いが混乱を招いていると言えるでしょう。

PS Tool では Format-Table/Format-List の出力を特別扱いし、独自に整形します。その優先順位はメニュー設定よりも高く、コンソール出力では省略されてしまった文字列も含めて展開するようにしています。Excel への出力結果がコンソールの結果と異なる場合には、Format-Table/Format-List を利用すると満足行く結果が得られます。

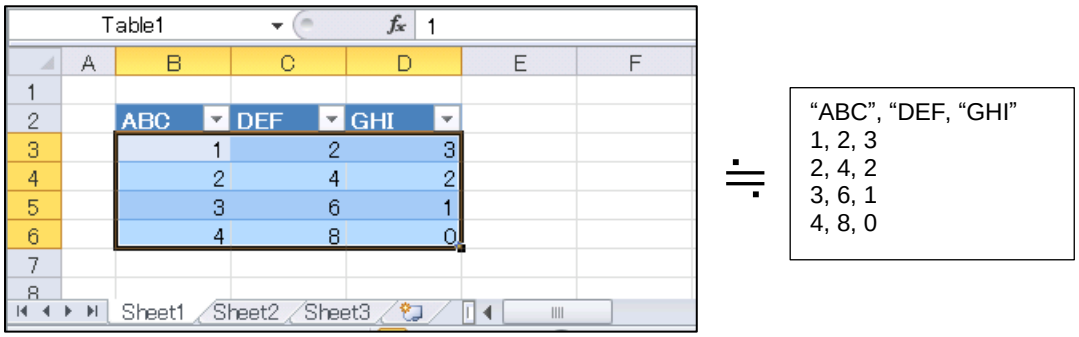


Excel のテーブルと Import-Csv

PowerShell では、Import-Csv コマンドレットで表形式データを入力データとして扱うことができます。Import-Csv コマンドレットから渡されたデータはオブジェクトの配列となっているので、 foreach で一行毎に読み込み、列名をプロパティとして指定することができます。(※スクリプト内であれば、ConvertTo-Csv コマンドレットで任意のオブジェクトを CSV ファイルのように扱うことができます)

参考：CSV ファイルを読み取る方法はありますか (Hey, Scripting Guy!)
http://gallery.technet.microsoft.com/office/c65d1c43-5204-46e2-9125-ba0af565bb38

PS Tool では任意の Excel テーブル（Excel 2003 までのリストを Import-Csv と同様の形式にデータ変換し PowerShell にパイプを通じて渡します。つまり、Import-Csv を使用することなしに表形式データを PowerShell の入力データとすることが可能です。



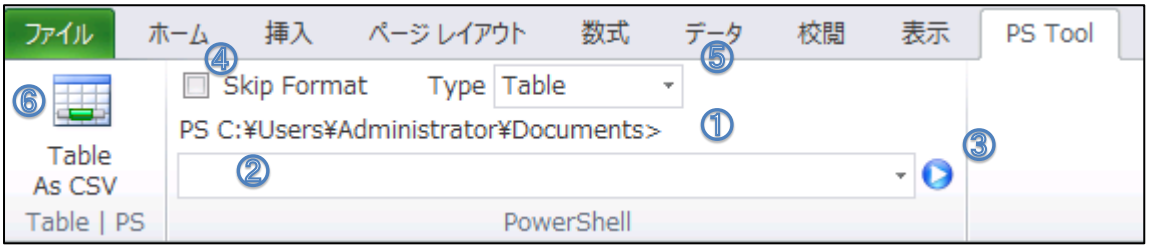
これにより次のような利点が生まれます。

①Worksheet 上に複数のテーブルを配置できる
②テーブルには独自のスタイルや、フィルター機能がある
③Excel の強力な編集機能、ワークシート関数を使える
④文字コード、データ型によるトラブルから開放される

PowerShell 3.0 からは Import-Csv にも –Encoding オプションが追加されているので便利になりましたが、PS Tool を使えば文字コードを意識する必要は殆どありません。Export-Csv/Import-Csv が不要になるわけではなく、PS Tool から呼び出すことでテーブルを CSV ファイル化したり、CSV ファイルをワークシートの任意の位置に読み込むことができます。

操作

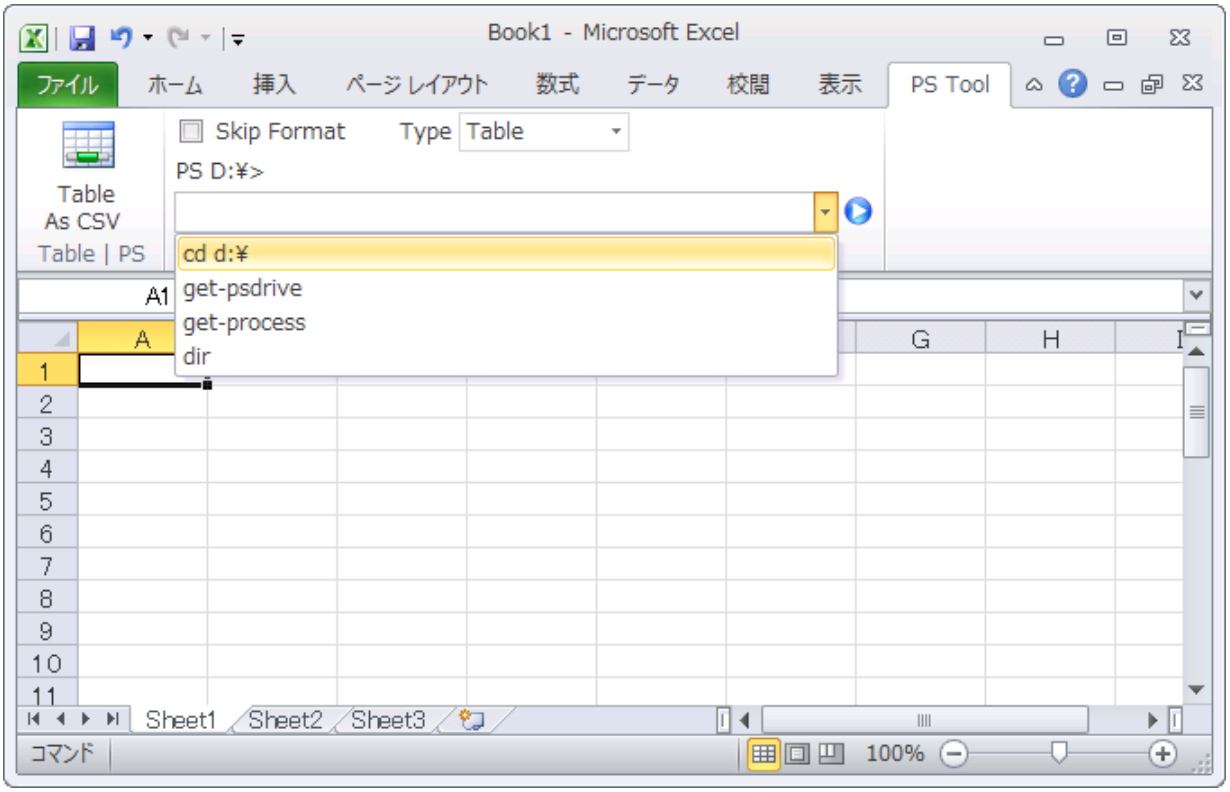
メニュー



①ディレクトリ位置ラベル
②コマンド入力ボックス
③コマンド実行ボタン
④書式無視チェックボックス
⑤出力形式選択
⑥テーブルの CSV 化ボタン

コマンドの実行

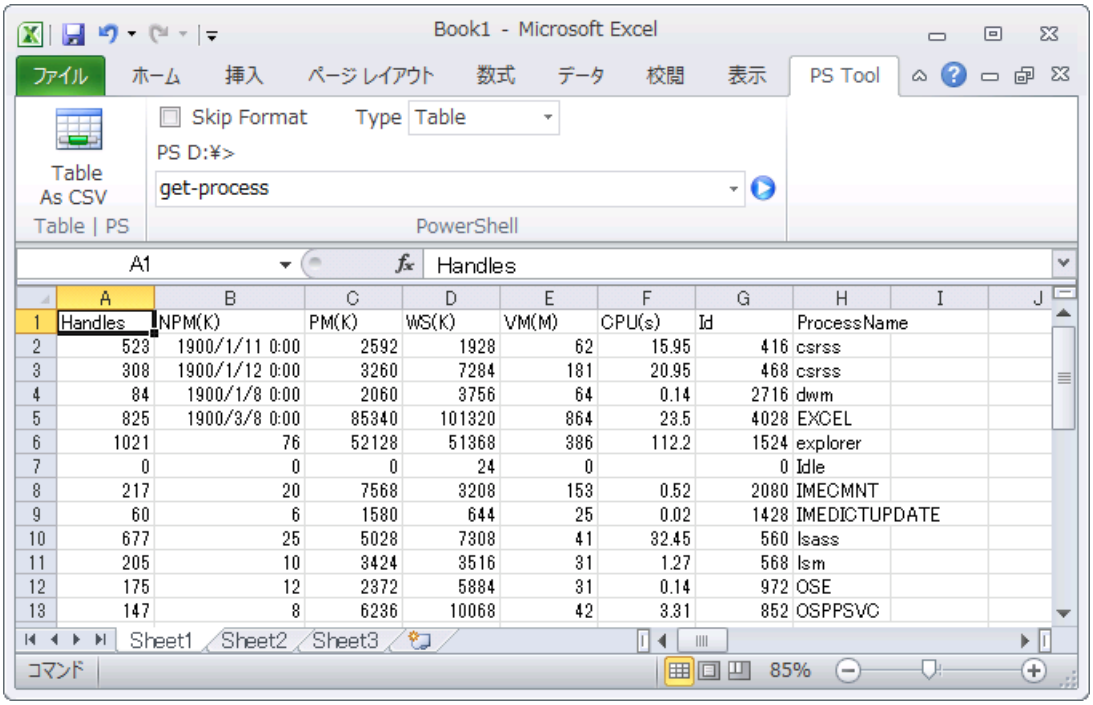
コマンド入力ボックスに任意の PowerShell コマンドを入力し、コマンド実行ボタンを押下します。コマンド入力ボックスはヒストリー機能が備わっているので、過去のコマンドに素早くアクセスできます。



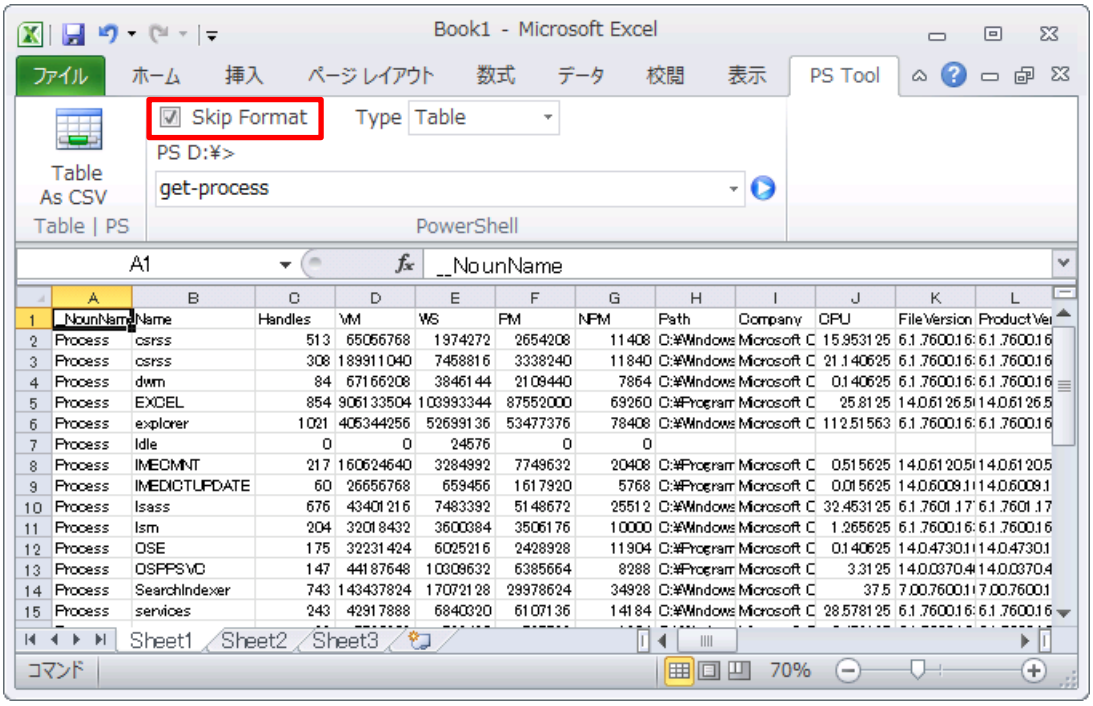
複数行に渡るコマンドはサポートしていないので、必要に応じてスクリプト化してください。

Skip Format(書式無視)

PowerShell では「書式ファイル」により暗黙の出力フィルターが設定されています。例えば Get-Process コマンドレットですが、



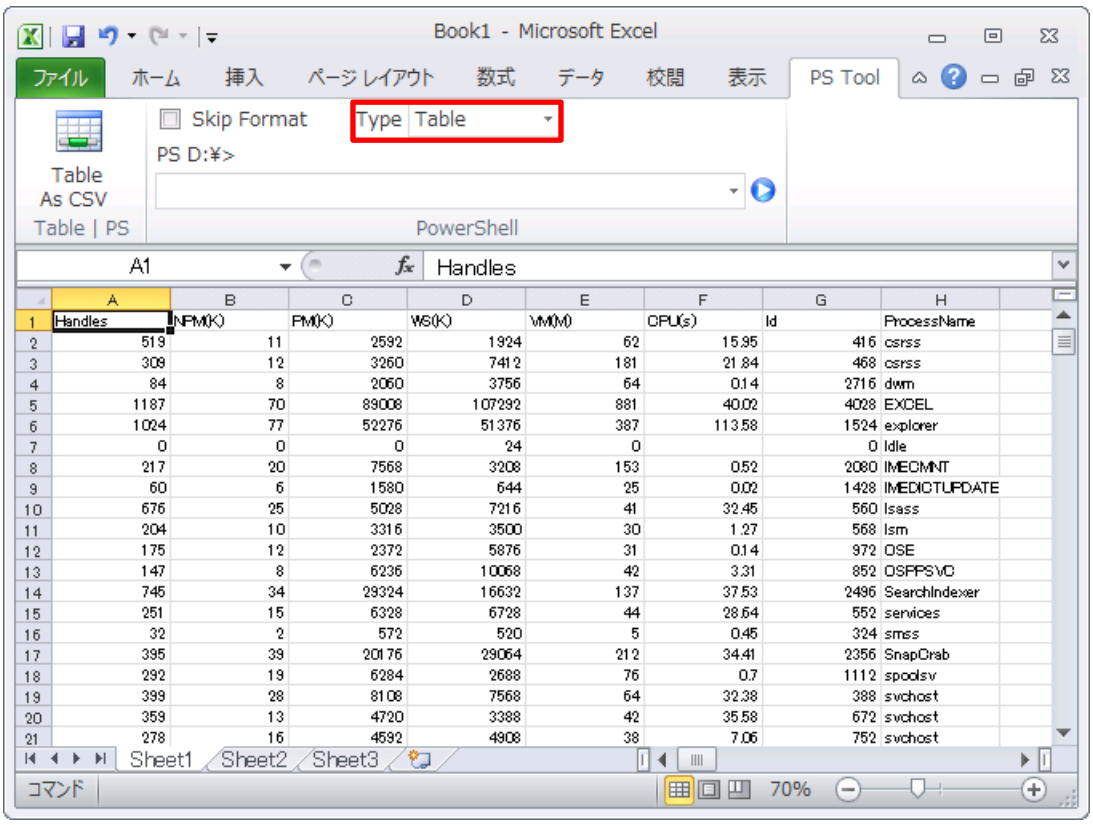
このように Handles, NPM(K), PM(K), WS(K), VM(M), CPU(s), Id, ProcessName という項目が表示されていますが、実際にはもっと多くのプロパティが隠されています。書式無視をチェックすることにより、全プロパティを表示するようになります。これは Format-Table -Property * とほぼ同様の結果となります。

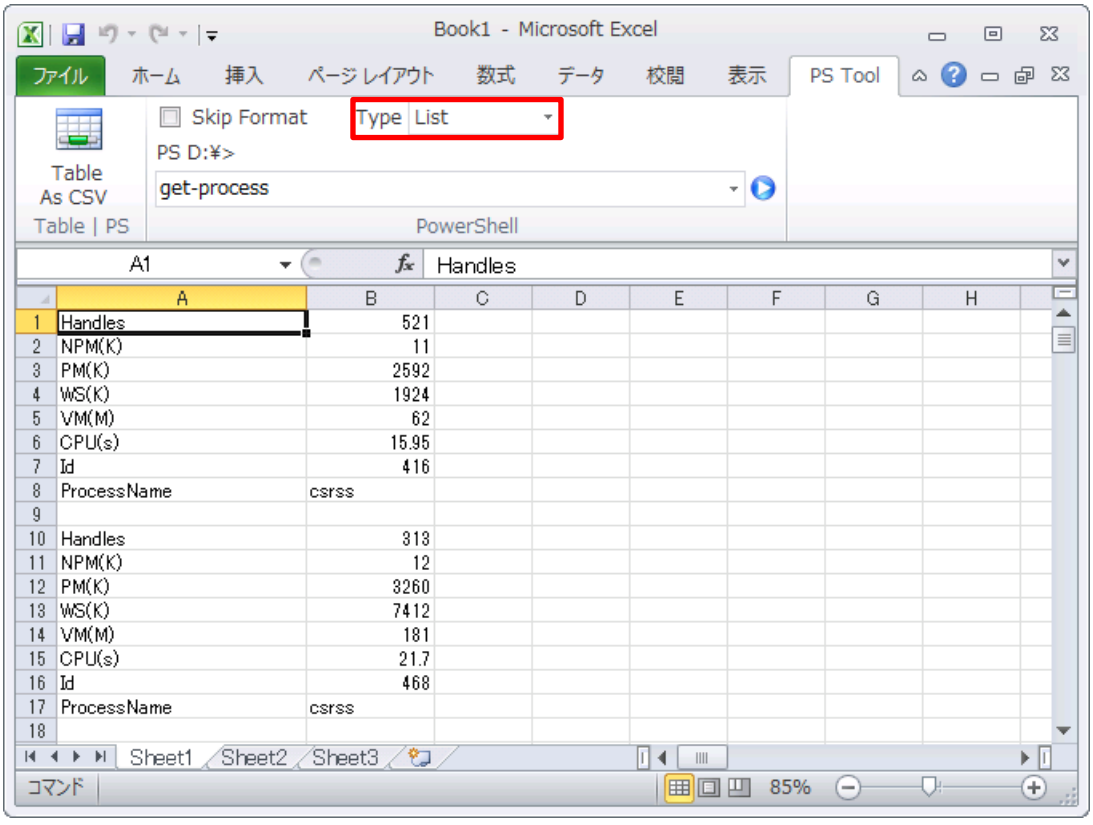


注意してみると、列名が微妙に変わっています。これは書式ファイルで列名を任意に変え、さらにスクリプトでプロパティ値を加工しているからです（(K)付きは 1024、(M)付きは 1048576 で割っています）。C:¥Windows¥System32¥WindowsPowerShell¥v1.0¥DotNetTypes.format.ps1xml を参照してください。

Type(出力形式)

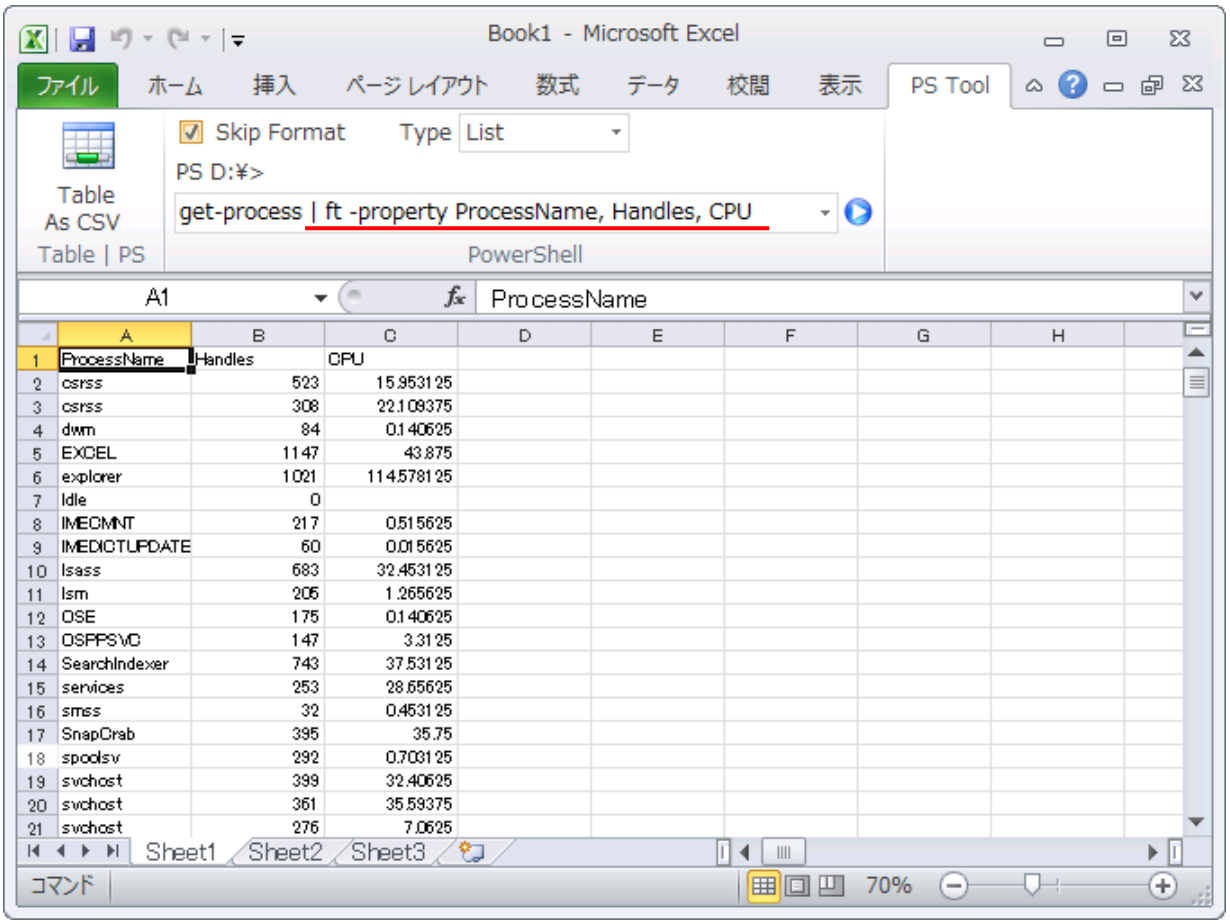
Excel では表形式での利用が圧倒的に便利ですが、単一のオブジェクトを表示したりする場合にはリスト形式が良い場合もあります。必要に応じて使い分けてください。





Format コマンドレットの利用

Format-Table(=ft) / Format-List(=fl)コマンドレットは特別です。これらが使われた場合、メニューの設定に優先して出力形式が決定されます。プロパティの順番を制御したり、PowerShell コンソールと同様の出力を得たい場合に使います。

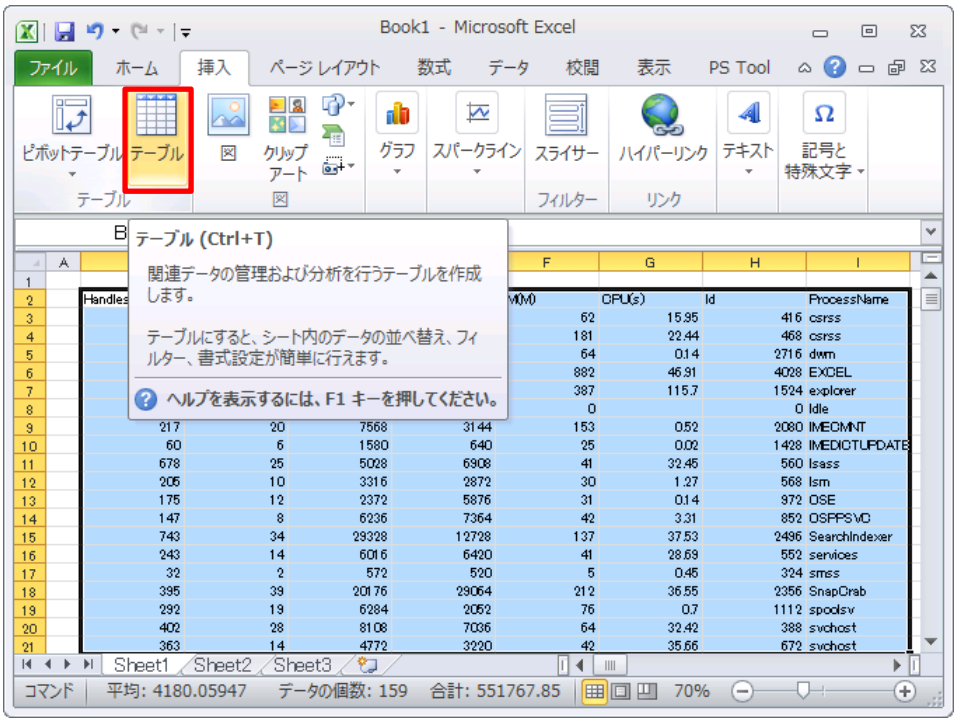


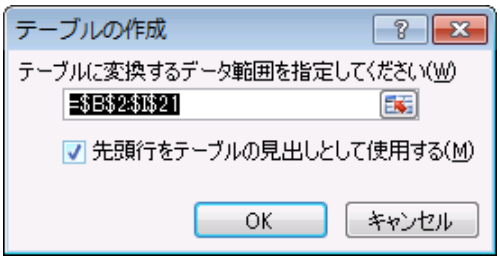
	A	B	C
1	ProcessName	Handles	CPU
2	csrss	523	15.953125
3	csrss	308	22.109375
4	dwm	84	0.140625
5	EXCEL	1147	43.875
6	explorer	1021	114.578125
7	Idle	0	
8	IMECMNT	217	0.515625
9	IMEDICTUPDATE	60	0.015625
10	lsass	683	32.453125
11	lsm	205	1.265625
12	OSE	175	0.140625
13	OSPPSVC	147	3.3125
14	SearchIndexer	743	37.53125
15	services	253	28.65625
16	smss	32	0.453125
17	SnapCrab	395	35.75
18	spoolsv	292	0.703125
19	svchost	399	32.40625
20	svchost	361	35.59375
21	svchost	276	7.0625

この例では、書式無視とリスト形式での出力を指定していますが、ft –property で指定された項目が優先して出力されていることがわかります。ここで、ft –property “CPU(s)” としてやってもうまくいきません。これはプロセス情報として “CPU(s)” というプロパティが存在しないからです。PowerShell のコンソールに出てくる列名は必ずしもプロパティ名と一致しないと認識してください。

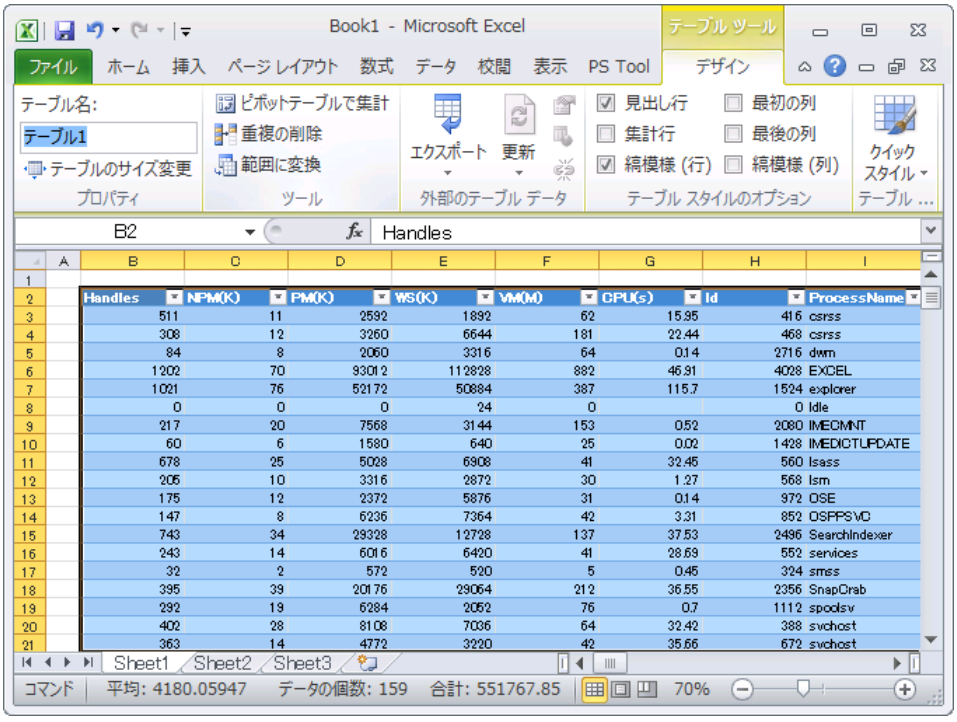
テーブル化

PS Tool ではコマンドレット出力を自動的にセルに展開するので、それをテーブル化するのは簡単です。テーブル化したい範囲を選択して、”挿入”-“テーブル” を選びます。



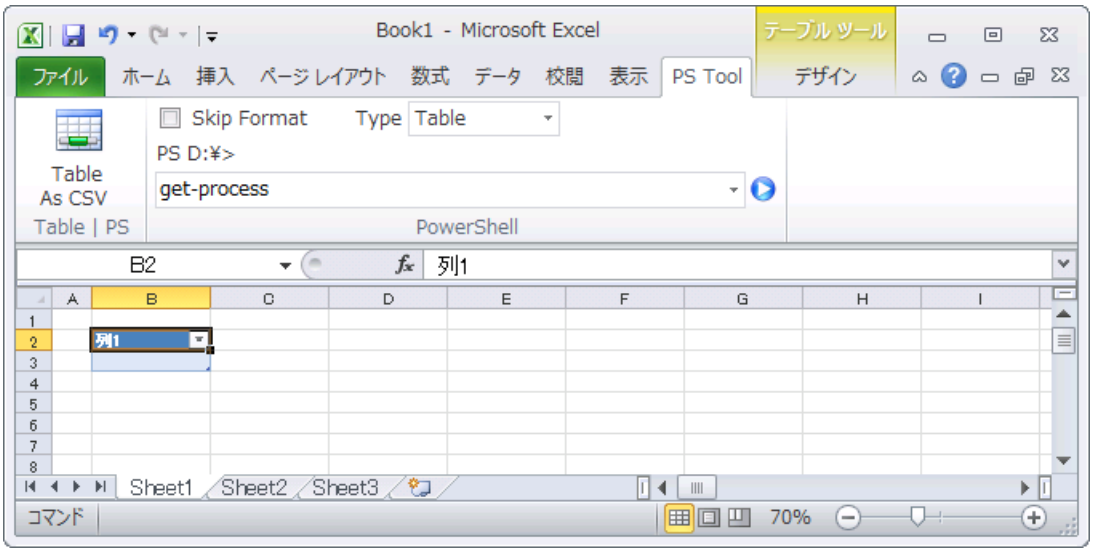


先頭行をテーブルの見出しにすると便利です。

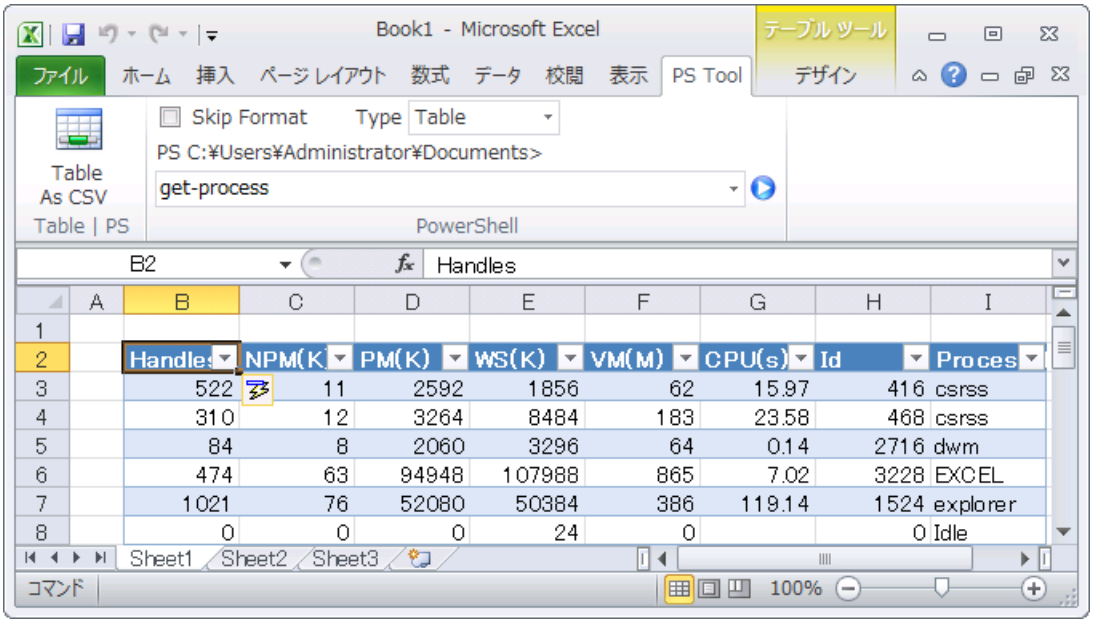


これでテーブル化の完了です。

予め空のテーブルを用意し、そこに PS Tool の出力を合わせるという方法もあります。

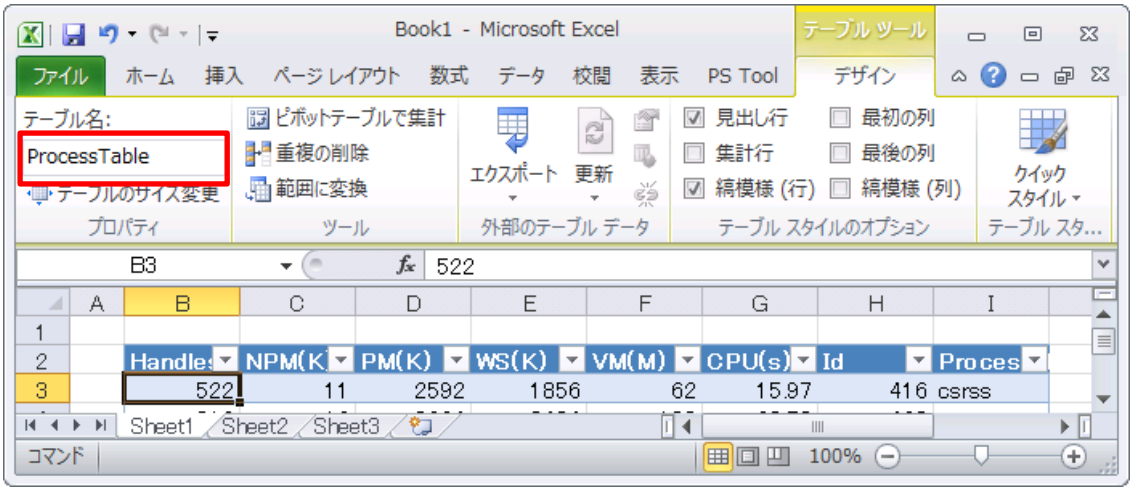


“列 1”の位置で、PS Tool からコマンドを実行すると、

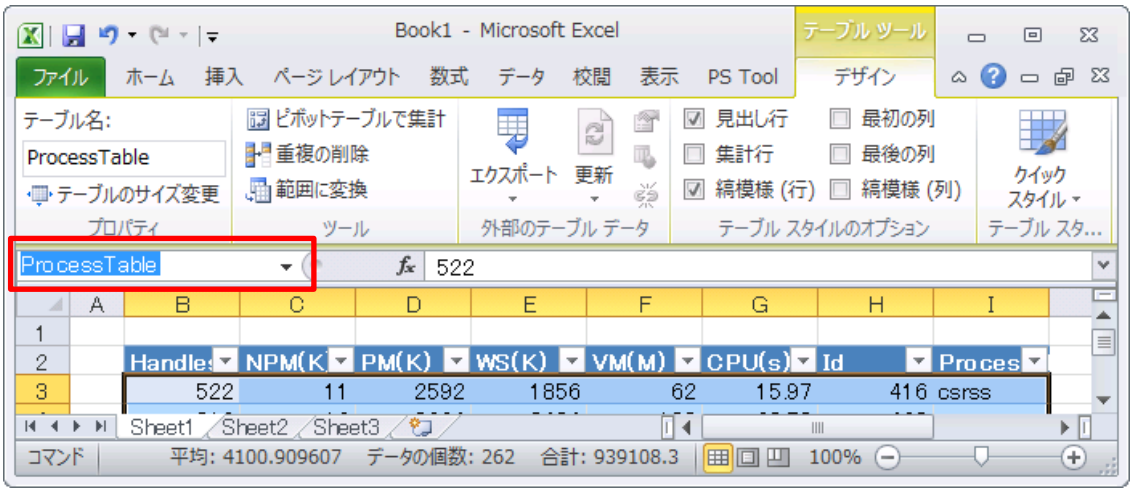


テーブル範囲が自動的に拡張されます。※Excel は、テーブルに隣接した範囲で編集を行うと自動的にテーブル範囲を拡張します。

テーブルを作成時には意味のあるテーブル名を付けることが重要です。テーブル名の変更は、テーブル選択時に現れる “テーブルツール・デザイン”-“テーブル名:” で行います。



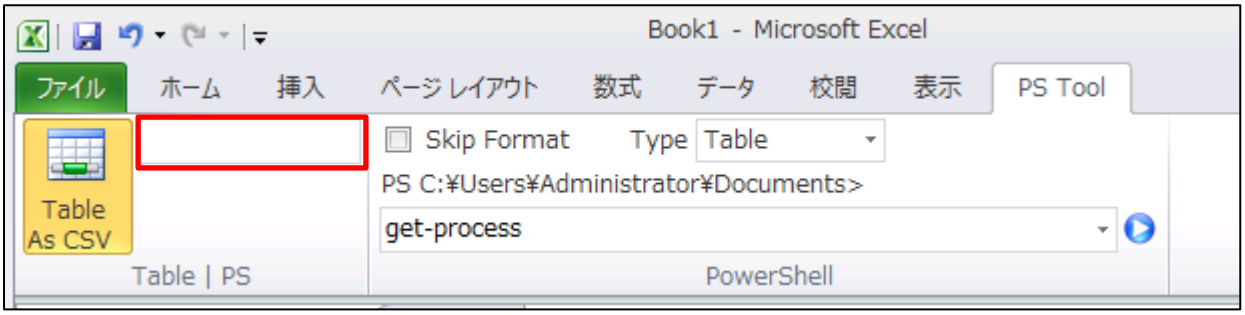
”名前ボックス” でテーブル名を変更しようとする場面を何度か見かけました。



名前ボックスでテーブルの選択は可能ですが、そこで名前を入力しても “テーブルのデータ範囲に名前を付ける” 事になり、テーブル本来の機能が使えません。テーブルには列情報、行情報、集計行、フィルター、スタイルといった付加価値が有り、名前付き範囲では代わりにはならないので注意してください。

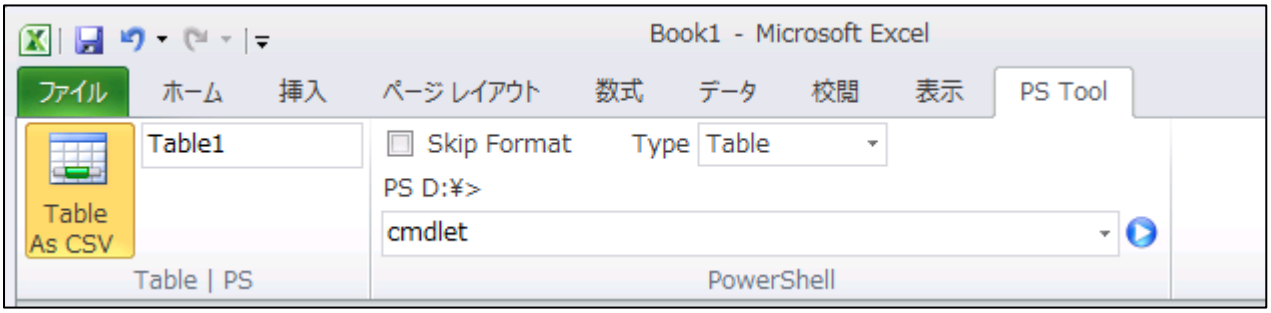
パイプ

“Table As CSV” ボタンはトグルになっています。押下時にはテーブル名を指定する TextBox が現れます。



ワークブック上に存在するテーブルが指定されている場合、そのテーブル情報を Import-Csv コマンドレットと互換性のあるデータに変換し、PowerShell に渡します。例えば、Excel 上の Table1 と d:¥hoge.csv が同様のデータを持つ場合、次の操作は同じ結果になります。

PS D:¥> Import-Csv d:¥hoge.csv | cmdlet



PS Tool では非表示にされた行に関しては変換しないので、不要な行を非表示にすることで余分な処理をスキップすることができます。これはテーブルのフィルターで非表示にされた行でも同じです。

Excel の強力な編集機能で管理されたテーブルをそのまま PowerShell で利用するのは非常に便利です。具体的には次章で運用例を示します。

運用例：ActiveDirectory の管理

ここでは架空のドメイン develop.con 上での ActiveDirectory ユーザーアカウントの保守作業を例にして PS Tool の運用例を説明します。ActiveDirectory 用のコマンドレット詳細については以下の URL を参照してください。

http://technet.microsoft.com/ja-jp/library/ee617195.aspx

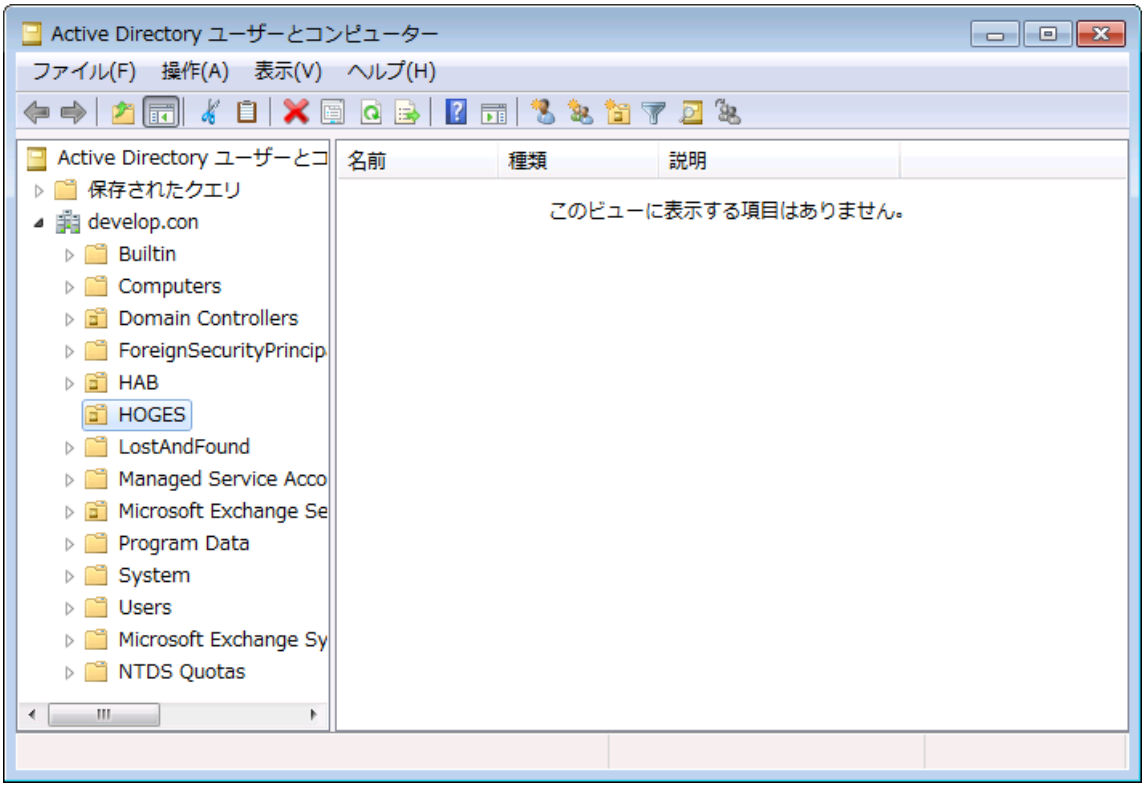
ActiveDirectory 標準の GUI は 1 アカウント毎の編集が基本なので、大量のアカウントを処理するためには様々なコマンドを駆使する必要がありました。PS Tool では Excel のワークシートをそのまま ActiveDirectory の管理画面として使えるので効率の良い運用が可能となります。

※この運用例のサンプルは、インストールファイルに含まれていますが、実際に試す場合には PowerShell で ActiveDirectory を管理するための環境が必要です。

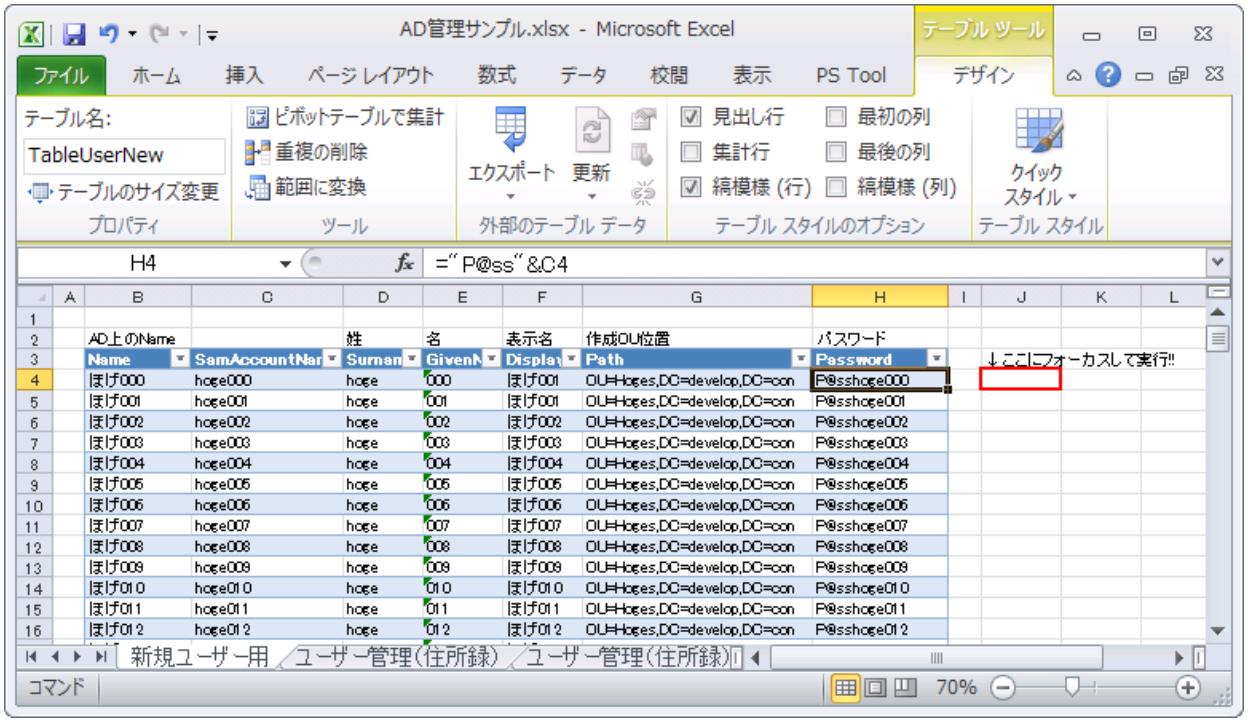
ユーザーアカウントの作成

新規アカウントを一つずつ入力するのは実際的ではありません。csvde コマンドや New-ADUser コマンドレットで一括登録するわけですが、どちらにしても入力となるアカウントデータを用意しなければなりません。もちろん PS Tool でもアカウントデータは必要ですが、面倒なファイル変換が不要になります。

ここでは develop.con の AD 上に専用の OU “HOGES” を用意し、そこに新規のユーザーアカウントをパスワード付きで作成するとします。



新規アカウント用のテーブル TableUserNew を用意します。



ここでは Name(=AD 上の名前), SamAccountName(=ログイン ID), Surname(=姓), GivenName(=名), DisplayName(=表示名), Path(=作成位置), Password(=パスワード) を用意します。 ※パスワードの発生は式を使って “P@ss”+ SamAccountName を発生させています。面倒な OU の LDAP 識別名も Excel なら簡単に編集できます。

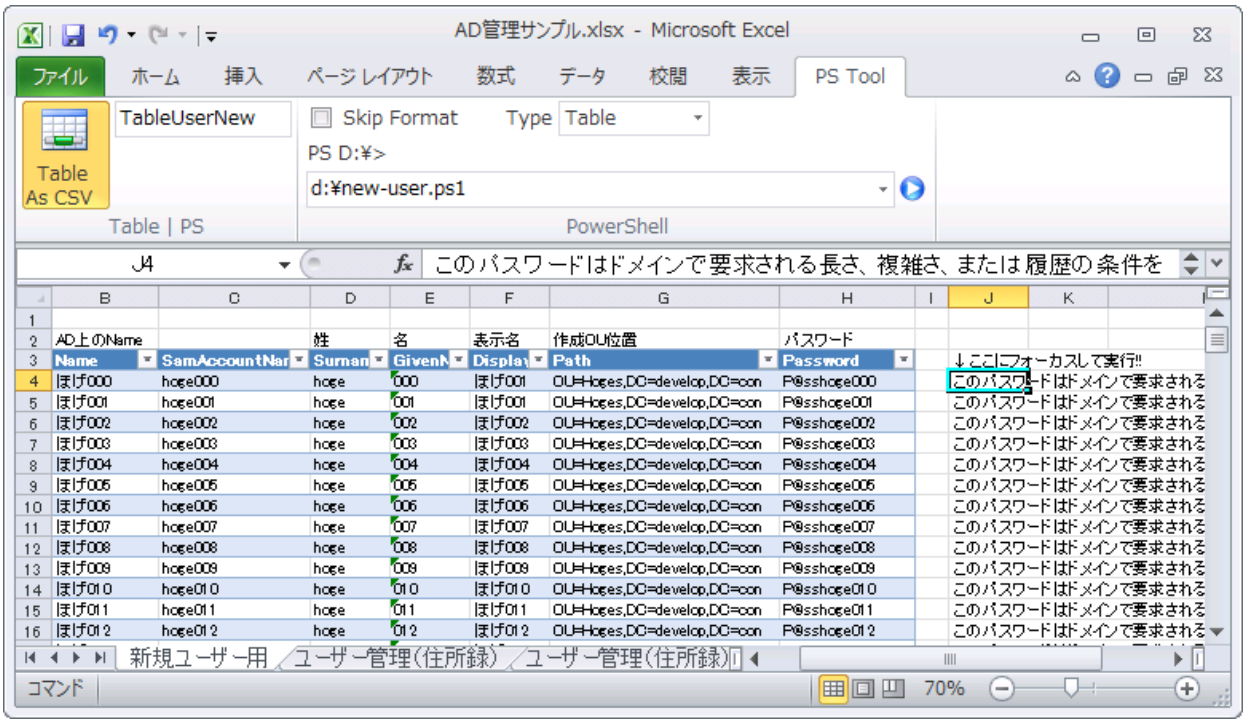
新規アカウント作成用スクリプト”New-User.ps1”を用意します。

```
# Get-ADUser にて ADUserObject Import-Csv 形式の情報がパイプで流れてくるので、それを順次処理する
# $input はパイプを示す予約変数
$input | foreach {
	try {
		# New-ADUser を使って必要な Property を設定していく
		# -PassThru で ADUser Object を拾う -ErrorAction は try-catch に必要
		$res = New-ADUser -name $_.Name -SamAccountName $_.SamAccountName `
			-Surname $_.Surname -GivenName $_.GivenName -DisplayName $_.DisplayName `
			-Path $_.Path `
			-PassThru -ErrorAction stop

		# パスワードの設定
		$res | Set-ADAccountPassword -Reset `
			-NewPassword (ConvertTo-SecureString -AsPlainText $_.Password –Force)`
			-ErrorAction stop

		# アカウントの有効化
		$res | Enable-ADAccount -ErrorAction stop

		"OK"
	}
	catch {
		$Error[0].ToString()
	}
}
```

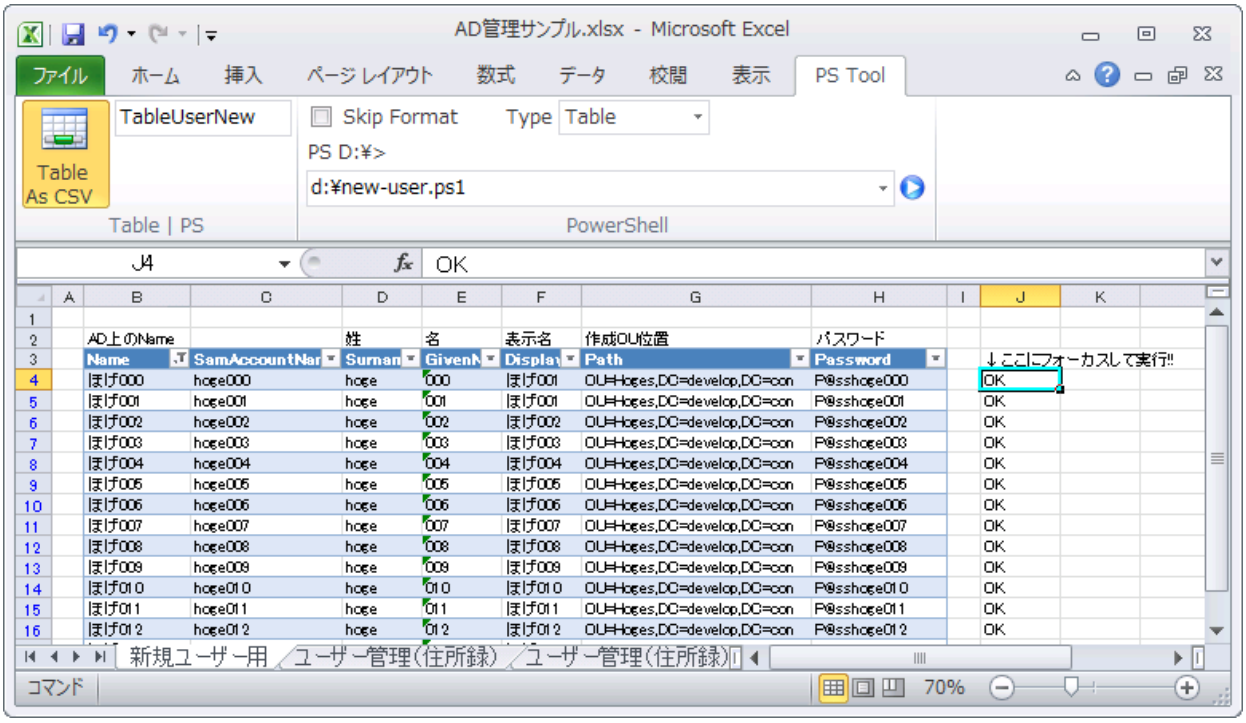
パイプ変数$input を foreach で回し、New-ADUser, Set-ADAccountPassword, Enable-ADAccount コマンドレットを順次適用しています。テーブルの各列名が foreach 内の “$_.列名”で参照できることに注目してください。オプション名と列名を合わせることで見通しの良いスクリプトになります。

テーブルの一行が１アカウントに対応し、foreach のループごとに処理されますが、処理毎に正常終了なら”OK”、何らかのエラーが発生すればそのエラー内容を文字列として出力しています。これで文字列の配列がスクリプトの実行結果となり、PS Tool ではそれを各行に反映します。これは PS Tool のログ手法として利用してください。

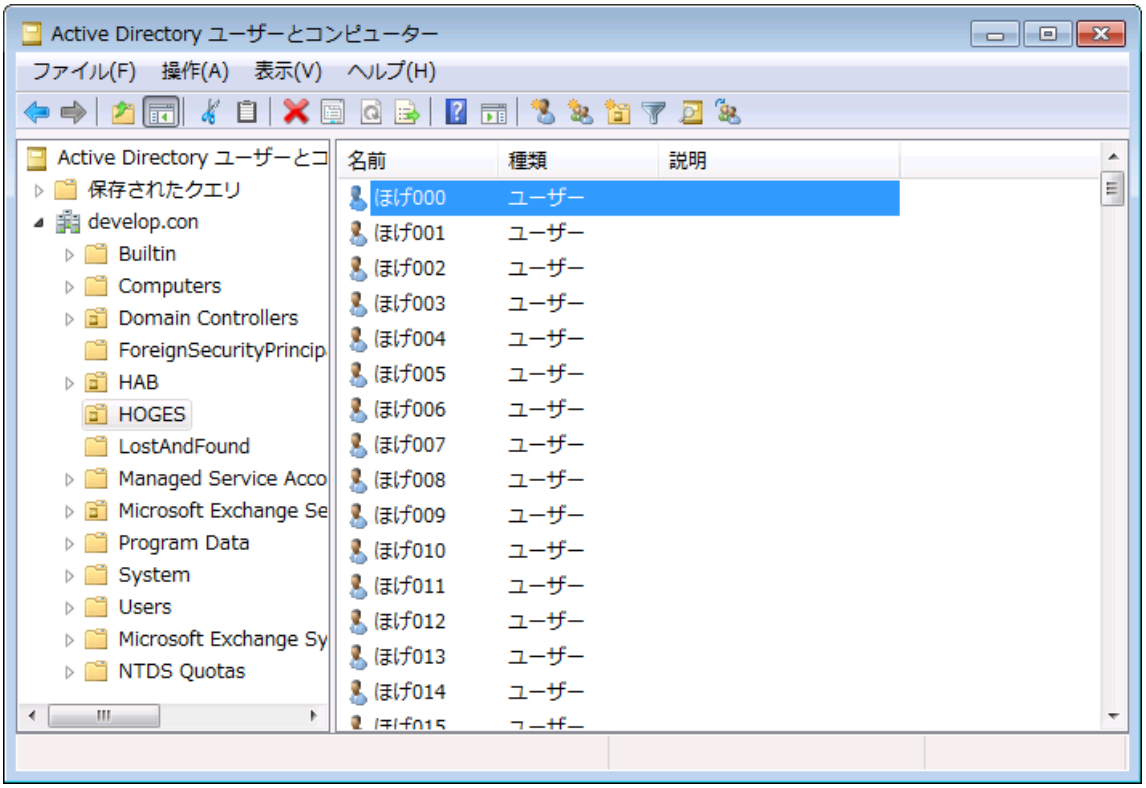
“Table AS CSV”ボタンを有効にし TableUserNew を指定して、赤で記したセルにフォーカスしてから new-user.ps1 を実行します。



パスワードが複雑さの要件を満たしていませんでした…orz　とりあえずセキュリティポリシーを変更してやり直します。※この辺りはドメインの状況により臨機応変に対処願います。



今度は無事にできました。



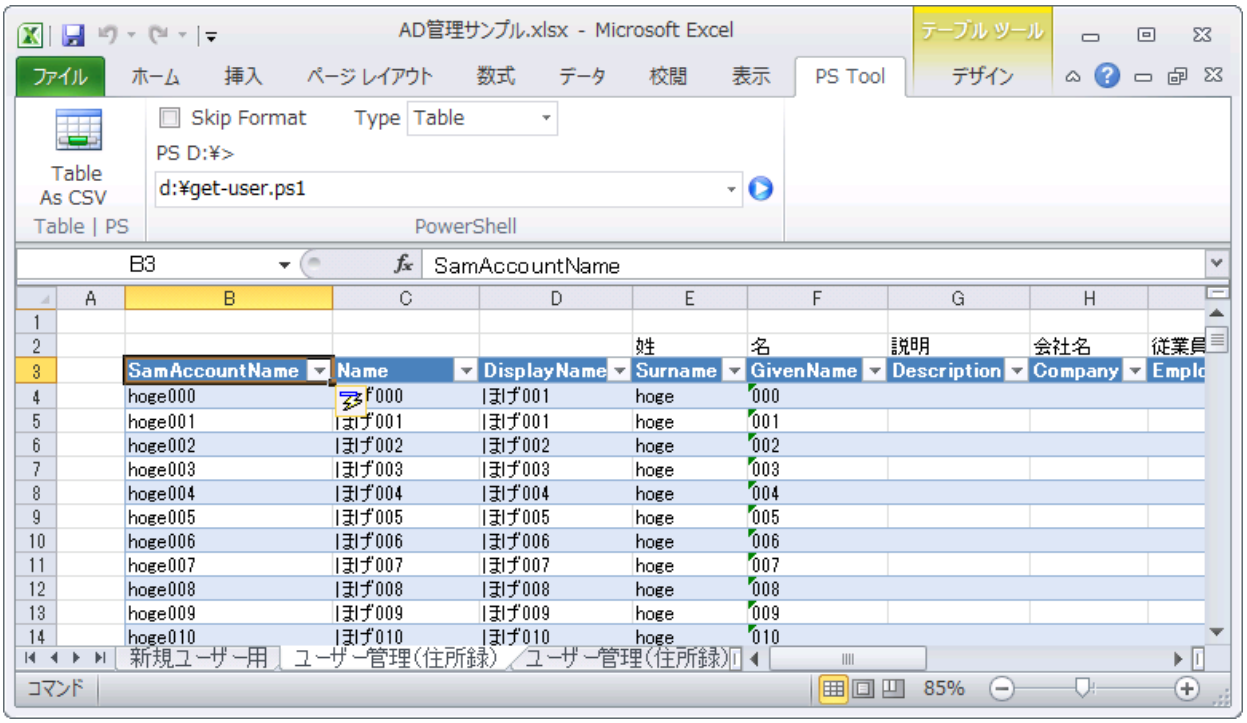
ユーザーアカウント情報の取得

前節では必要最低限の項目でアカウントを作成しましたが、実際には“必要な”情報を取得しなければなりません。ここではユーザー情報取得用のスクリプト”Get-User.ps1”を用意します。

```
# Get-ADUser のオプションは多彩。ここでは ”ほげ” で始まるアカウントの一覧を対象にする
# 全プロパティを取得後、Select-Object で必要な項目のみ抽出する
Get-ADUser -Filter { Name -like "ほげ*" } -Properties * |
        Select-Object SamAccountName,Name,DisplayName,Surname,GivenName,Description, `
                Company,EmployeeNumber,Department,Title, `
                HomePhone,MobilePhone,OfficePhone, `
                PostalCode,State,City,StreetAddress
```

Get-ADUser で必要なユーザーの全プロパティを取得し、Select-Object で必要な項目を選択しているだけです。今回は AD 上で住所録に使えそうな項目を選んでみました。 AD には数多くの項目があるので、必要に応じて編集してください。

Get-User.ps1 を実行し、作成される結果を TableUser と名前を付けます。

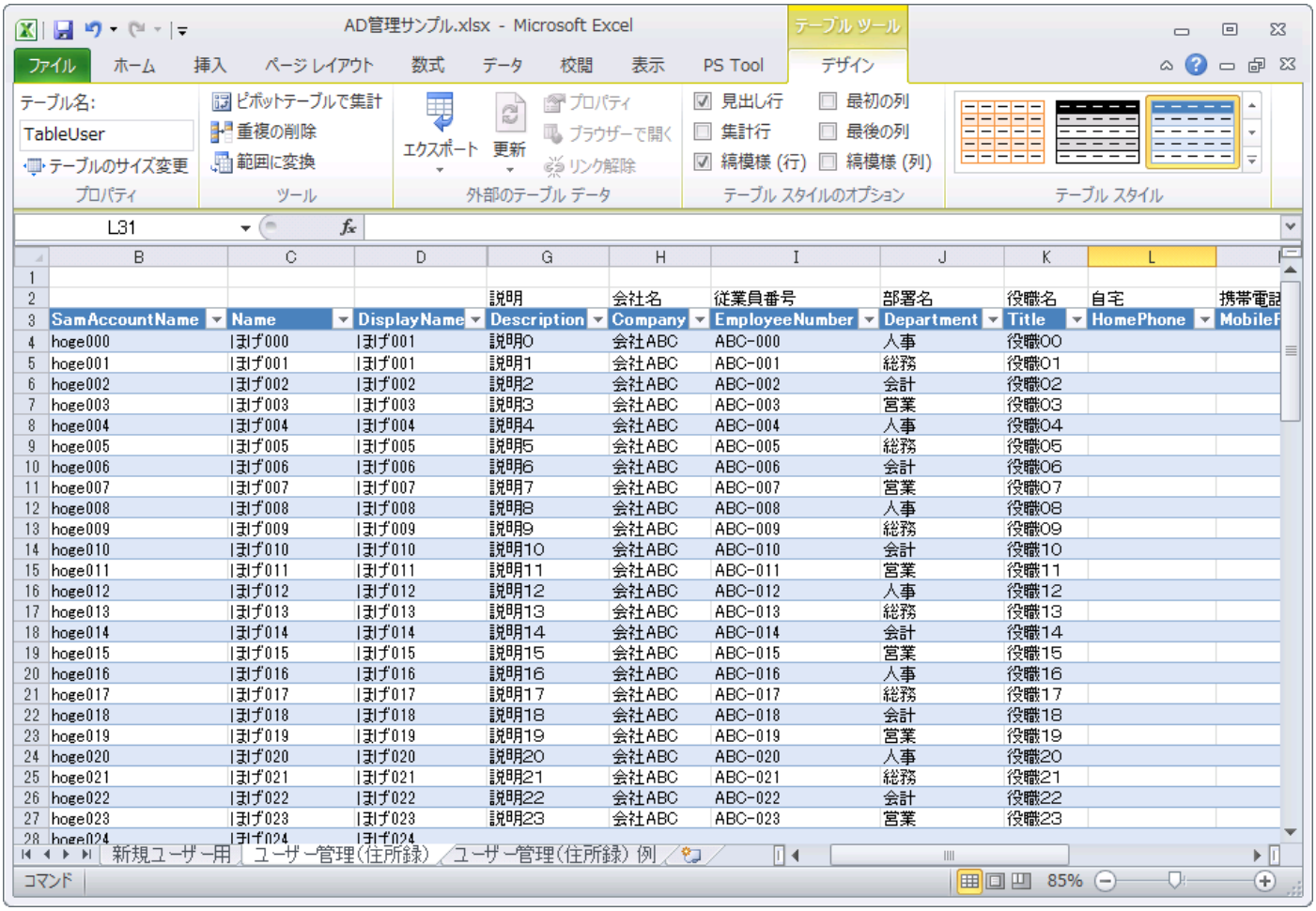


AD の列名は英語なので、Excel 上では日本語の説明等を併記すれば親切です。

作られたばかりのアカウントなので、ほとんどの項目が空白です。現場ではこれらの空白項目を素早く埋めることが任務になります。

ユーザーアカウント情報の編集

必要とされる項目を編集するのは Excel の得意とする分野です。各セルでは任意のワークシート関数や数式を使うことが可能です。



	B	C	D	G	H	I	J	K	L
2				説明	会社名	従業員番号	部署名	役職名	自宅
3	SamAccountName	Name	DisplayName	Description	Company	EmployeeNumber	Department	Title	HomePhone
4	hoge000	ほげ000	ほげ001	説明0	会社ABC	ABC-000	人事	役職00	
5	hoge001	ほげ001	ほげ001	説明1	会社ABC	ABC-001	総務	役職01	
6	hoge002	ほげ002	ほげ002	説明2	会社ABC	ABC-002	会計	役職02	
7	hoge003	ほげ003	ほげ003	説明3	会社ABC	ABC-003	営業	役職03	
8	hoge004	ほげ004	ほげ004	説明4	会社ABC	ABC-004	人事	役職04	
9	hoge005	ほげ005	ほげ005	説明5	会社ABC	ABC-005	総務	役職05	
10	hoge006	ほげ006	ほげ006	説明6	会社ABC	ABC-006	会計	役職06	
11	hoge007	ほげ007	ほげ007	説明7	会社ABC	ABC-007	営業	役職07	
12	hoge008	ほげ008	ほげ008	説明8	会社ABC	ABC-008	人事	役職08	
13	hoge009	ほげ009	ほげ009	説明9	会社ABC	ABC-009	総務	役職09	
14	hoge010	ほげ010	ほげ010	説明10	会社ABC	ABC-010	会計	役職10	
15	hoge011	ほげ011	ほげ011	説明11	会社ABC	ABC-011	営業	役職11	
16	hoge012	ほげ012	ほげ012	説明12	会社ABC	ABC-012	人事	役職12	
17	hoge013	ほげ013	ほげ013	説明13	会社ABC	ABC-013	総務	役職13	
18	hoge014	ほげ014	ほげ014	説明14	会社ABC	ABC-014	会計	役職14	
19	hoge015	ほげ015	ほげ015	説明15	会社ABC	ABC-015	営業	役職15	
20	hoge016	ほげ016	ほげ016	説明16	会社ABC	ABC-016	人事	役職16	
21	hoge017	ほげ017	ほげ017	説明17	会社ABC	ABC-017	総務	役職17	
22	hoge018	ほげ018	ほげ018	説明18	会社ABC	ABC-018	会計	役職18	
23	hoge019	ほげ019	ほげ019	説明19	会社ABC	ABC-019	営業	役職19	
24	hoge020	ほげ020	ほげ020	説明20	会社ABC	ABC-020	人事	役職20	
25	hoge021	ほげ021	ほげ021	説明21	会社ABC	ABC-021	総務	役職21	
26	hoge022	ほげ022	ほげ022	説明22	会社ABC	ABC-022	会計	役職22	
27	hoge023	ほげ023	ほげ023	説明23	会社ABC	ABC-023	営業	役職23	

編集に関しては多くを語らなくても、現場には Excel ユーザーがいるから大丈夫です。PS Tool の最大の利点は Excel で編集できることに他なりません。

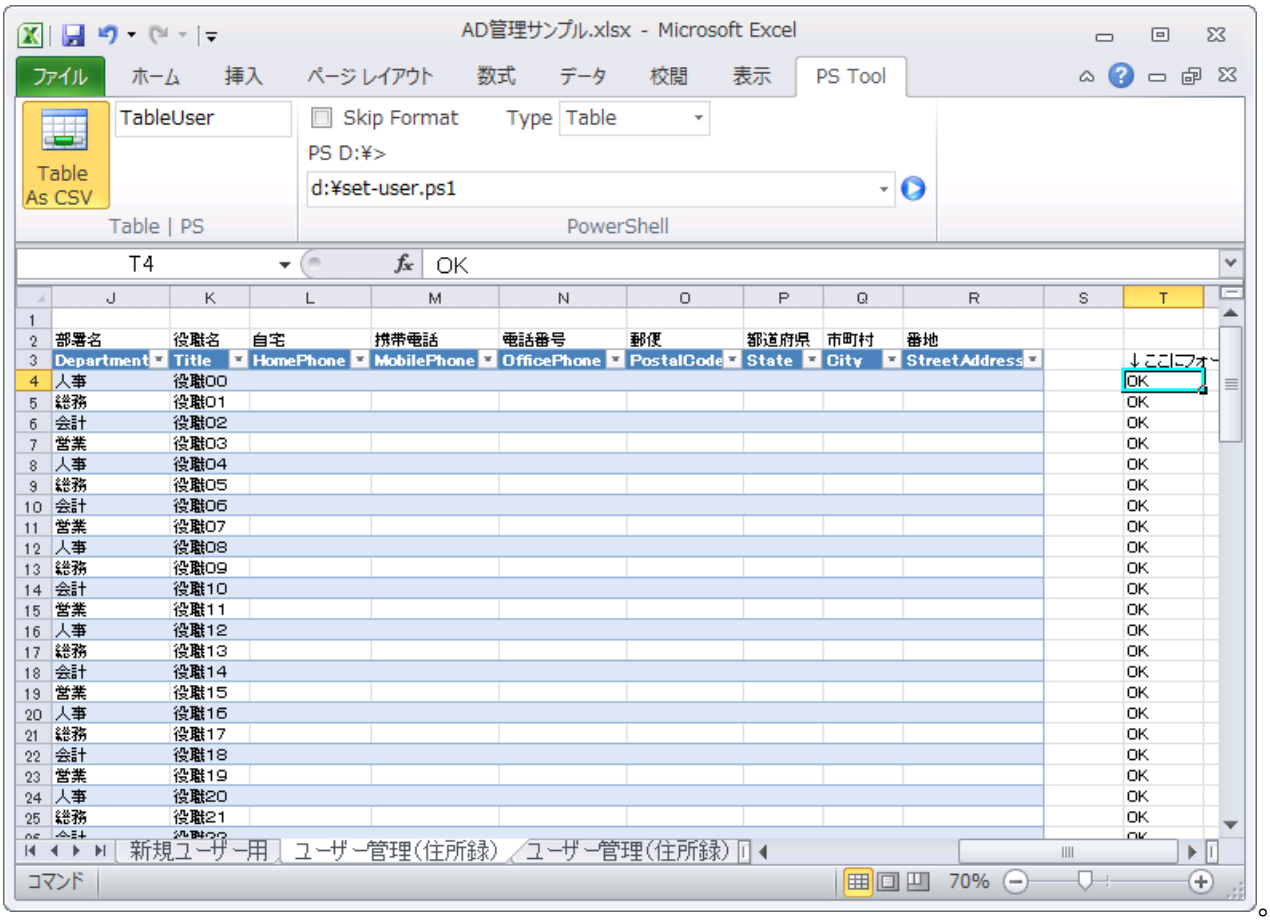
ユーザーアカウント情報の更新

TableAdUser を指定し、”Set-User.ps1”を実行します。

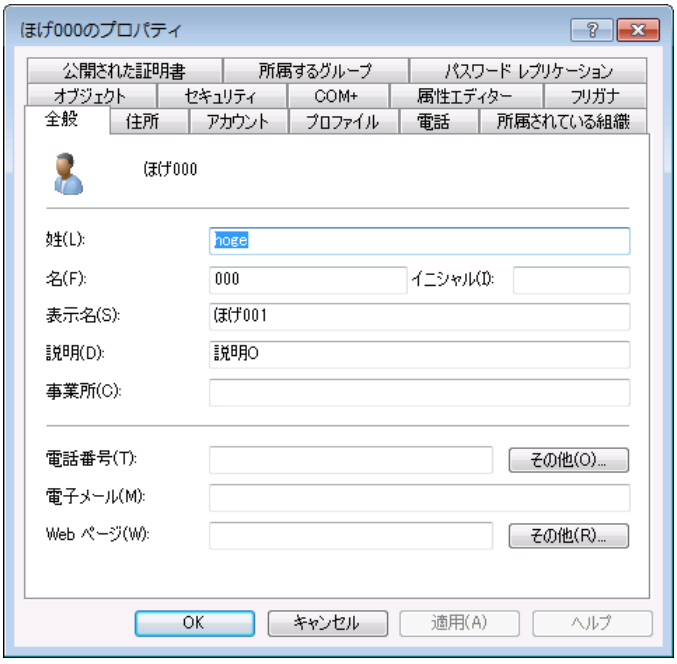
```
# 設定は Get-ADUser と Set-ADUser を組み合わせる。
# これはオプションでは対応できないプロパティに対処するためである。

$input | foreach {
	try {
		# SamAccountName を ID として全プロパティを取得
		$user = Get-ADUser -Identity $_.SamAccountName -Properties * -ErrorAction Stop

		# Propert y の設定…力技だけど無難
		# $user.Name = $_.Name                                  # Name の変更は Rename-ADObject
		$user.DisplayName = $_.DisplayName                      # 表示名
		$user.Surname = $_.Surname                              # 姓
		$user.GivenName = $_.GivenName                          # 名
		$user.Description = $_.Description                      # 説明
		$user.Company = $_.Company                              # 会社名
		$user.EmployeeNumber = $_.EmployeeNumber                # 社員番号
		$user.Department = $_.Department                        # 部署
		$user.Title = $_.Title                                  # 役職
		$user.HomePhone = $_.HomePhone
		$user.MobilePhone = $_.MobilePhone
		$user.OfficePhone = $_.OfficePhone
		$user.PostalCode = $_.PostalCode                        # 郵便番号
		$user.State = $_.State                                  # 都道府県
		$user.City = $_.City                                    # 市町村
		$user.StreetAddress = $_.StreetAddress                  # 住所
#		$user.xxx = $_.xxx
#		$user.xxx = $_.xxx
#		$user.xxx = $_.xxx

		Set-ADUser -Instanc $user -ErrorAction Stop
		"OK"
	}
	catch {
		$Error[0].ToString()
	}
}
```

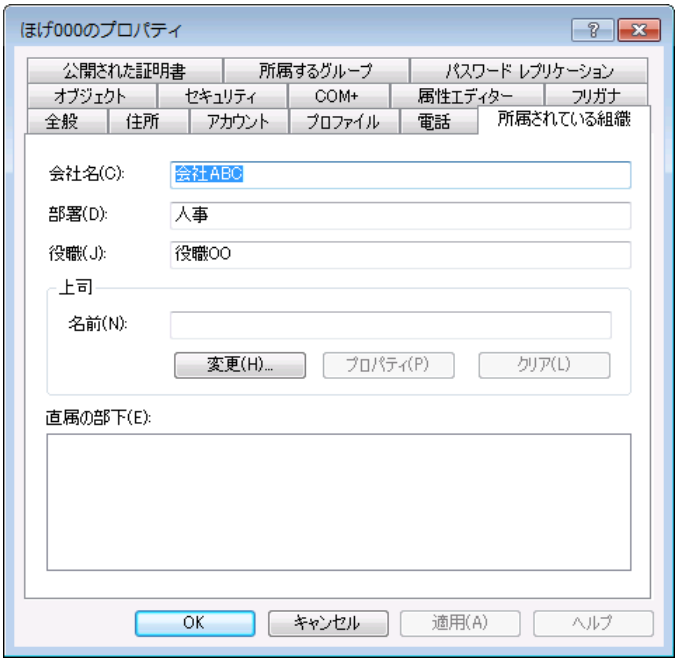
Set-User.ps1 も、パイプ変数$input を利用して一行毎に AD の更新をしているだけです。一旦 Get-ADUser で ADUser オブジェクトを取得し、そのインスタンスに変更を加え Set-ADUser に渡しています。少々強引ですが、１コマンドで長いオプションを記述するよりこちらが好みです。

実行してみます



AD 上で確認してみます。





Excel 上のデータが AD に反映しています。

スクリプトの壁はありますが、Excel 上で ActiveDirectory の管理が可能だとわかります。

あとがき

PS Tool は、Excel でサーバー管理ツールを構築していた際に、サーバー情報を簡単に収集するツールから生まれました。初めは特定コマンドだけの対応でしたが、PowerShell の基本 Object である PSObject を調査している間に応用範囲が広がり、現在の形になりました。 PowerShell と Excel の融合によって、Excel から.Net Framework を使うことが可能となり、多くのデータを簡単に Excel に取り込めるようになったと思っています。

Microsoft はサーバー製品に対して管理用 PowerShell コマンドレットを充実させているので、Exchange、SharePoint といった代表的なサーバー製品は PS Tool で管理する対象になるでしょう。PowerShell は Windows に必要不可欠な要素になっています。PS Tool がその普及に役立てば幸いです。

運用例では単純 AD 管理の例を紹介しましたが、実際の運用では業務に即したカスタマイズが必要になります。例えば、

① “Action” という列を新設してその内容によってスクリプトの動作を変えるようする
② SharePoint, Exchange, Hyper-V 等と連動させる
③ 専用のアイコンを追加してスクリプトの打ち込みを省略する

①のアイディアは PS Tool の応用としては簡単なのでトライしてみてください。②や③は業務によって難易度が変わってきます。PowerShell スクリプトを用意すれば、Excel で管理が可能になる分野は非常に多いので、興味がある方はご相談ください。

URL: http://manamana.ddo.jp/blog/page/pstool.aspx
E-mail: manabe@manamana.ddo.jp

真鍋正幸（まなべ まさゆき）

既知の不具合・制限

コマンドの実行を停止できない

PowerShell のコマンドが現行のワークスペースに戻るまで Excel に制御が戻りません。極端な例ですが、PowerShell.exe を実行すると事実上固まったような状態になります。 Job としてコマンドを呼び出せば解決できると思っていましたが現在調査中です。

項目が配列の場合、出力が型名として出力される。

プロパティには様々な値が設定可能であり、配列を指定することも可能です。これはセルに階層的を配置することになりますが、現状では単純に型名を出力しています。PowerShell では配列を｛val1,val2…｝の様に記述しますが、val にもまた配列が指定可能であり…要するにキリがありません。コンソール出力と同じ結果を得たい場合には Format-Table を使用してください。

この制限は AD 管理でも問題になります。AD の GUI と同様の情報を得たい場合もあるので対応を考えています。

警告、エラーがセルに展開できない

PowerShell コンソールでは警告が黄色、エラーが赤色で出力されて便利ですが、PS Tool では実行時エラーのみをトラップしてダイアログ表示しています。必要に応じてファイルにリダイレクトするようにしてください。

PowerShell 3.0 からは警告に対しても、3>や 3>&1 のようにリダイレクトできるようになっているので助かります。

エスケープ文字

Exe 形式の実行ファイルの多くは文字列を出力します。PS Tool は文字列をサポートしていますが、改行コードが来たら単純に次の行に移る程度の事しかしていません。エスケープ文字を使った画面の制御には未対応なので、Chkdsk.exe の様に同一行に進行状態を表示するようなコマンドも複数行に渡る出力になります。