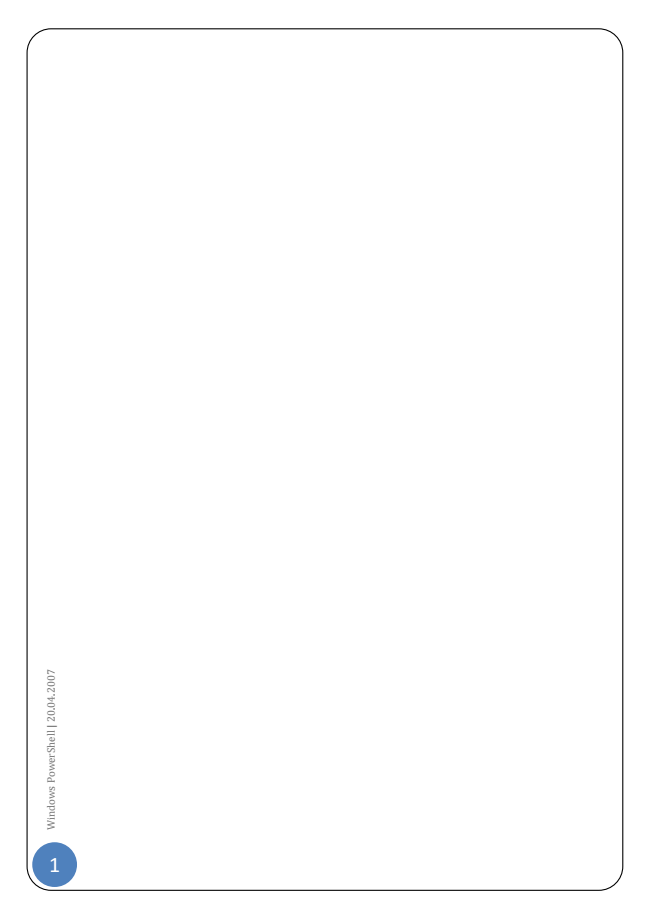


Windows PowerShell

*An introduction to scripting technologies for people with no real background knowledge*

**MICROSOFT SWITZERLAND**

20 April 2007 Frank Koch (BERN) Developer & Platform Evangelism



1

W

i

n

d

o

w

s

P

o

w

e

r

S

h

e

l

l

|

2

0

.

0

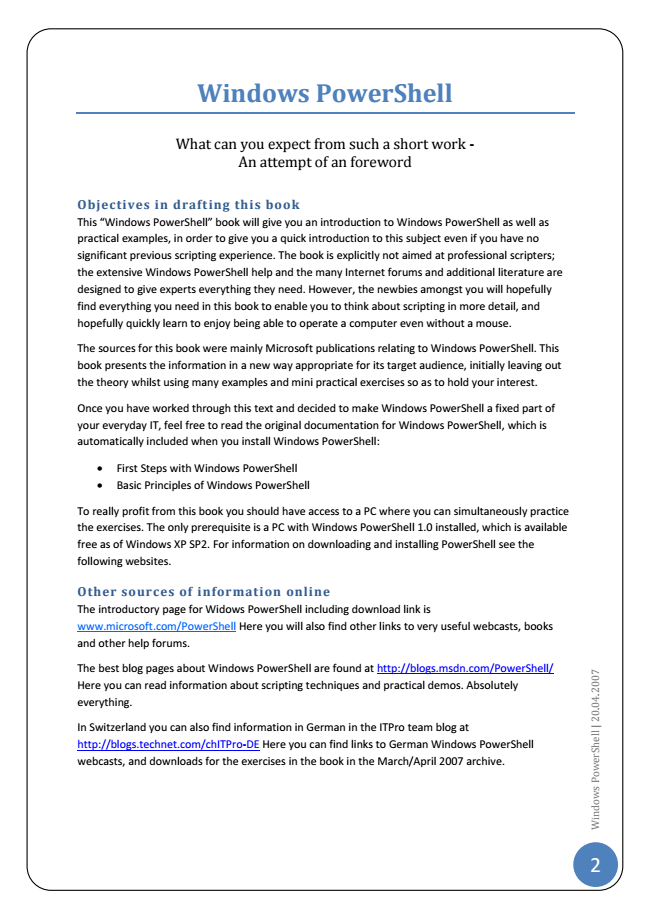
4

.2

0

0

7



**Windows PowerShell**

What can you expect from such a short work - An attempt of an foreword

Objectives in drafting this book This “Windows PowerShell” book will give you an introduction to Windows PowerShell as well as practical examples, in order to give you a quick introduction to this subject even if you have no significant previous scripting experience. The book is explicitly not aimed at professional scripters; the extensive Windows PowerShell help and the many Internet forums and additional literature are designed to give experts everything they need. However, the newbies amongst you will hopefully find everything you need in this book to enable you to think about scripting in more detail, and hopefully quickly learn to enjoy being able to operate a computer even without a mouse.

The sources for this book were mainly Microsoft publications relating to Windows PowerShell. This book presents the information in a new way appropriate for its target audience, initially leaving out the theory whilst using many examples and mini practical exercises so as to hold your interest.

Once you have worked through this text and decided to make Windows PowerShell a fixed part of your everyday IT, feel free to read the original documentation for Windows PowerShell, which is automatically included when you install Windows PowerShell:

• First Steps with Windows PowerShell

• Basic Principles of Windows PowerShell

To really profit from this book you should have access to a PC where you can simultaneously practice the exercises. The only prerequisite is a PC with Windows PowerShell 1.0 installed, which is available free as of Windows XP SP2. For information on downloading and installing PowerShell see the following websites.

Other sources of information online The introductory page for Widows PowerShell including download link is www.microsoft.com/PowerShell Here you will also find other links to very useful webcasts, books and other help forums.

The best blog pages about Windows PowerShell are found at http://blogs.msdn.com/PowerShell/ Here you can read information about scripting techniques and practical demos. Absolutely everything.

In Switzerland you can also find information in German in the ITPro team blog at http://blogs.technet.com/chITPro-DE Here you can find links to German Windows PowerShell webcasts, and downloads for the exercises in the book in the March/April 2007 archive.

2

W

i

n

d

o

w

s

P

o

w

e

r

S

h

e

l

l

|

2

0

.

0

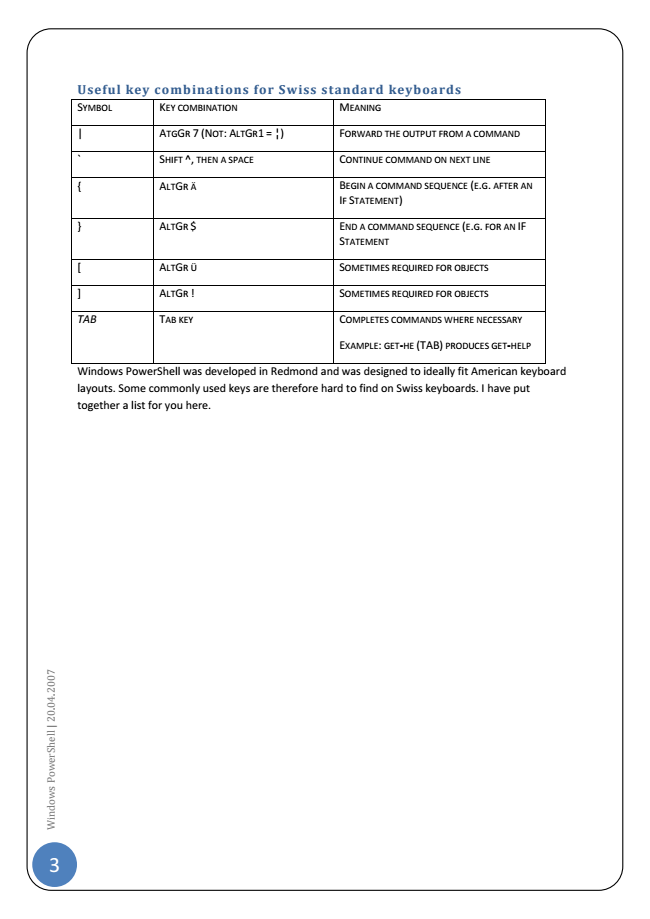
4

.2

0

0

7



3

W

i

n

d

o

w

s

P

o

w

e

r

S

h

e

l

l

|

2

0

.

0

4

.2

0

0

7

**Useful key combinations for Swiss standard keyboards SYMBOL KEY COMBINATION MEANING**

| ATGGR 7 (NOT: ALTGR1 = ¦) FORWARD THE OUTPUT FROM A COMMAND

` SHIFT ^, THEN A SPACE CONTINUE COMMAND ON NEXT LINE

{ ALTGR Ä BEGIN A COMMAND SEQUENCE (E.G. AFTER AN

IF STATEMENT)

} ALTGR $ END A COMMAND SEQUENCE (E.G. FOR AN IF

STATEMENT

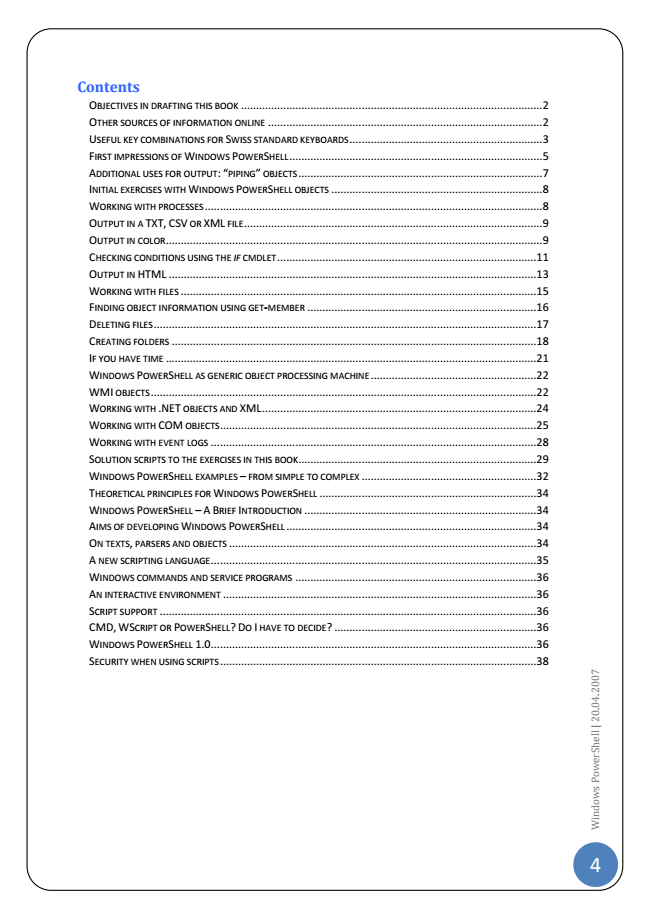
[ ALTGR Ü SOMETIMES REQUIRED FOR OBJECTS

] ALTGR ! SOMETIMES REQUIRED FOR OBJECTS

TAB TAB KEY COMPLETES COMMANDS WHERE NECESSARY

EXAMPLE: GET-HE (TAB) PRODUCES GET-HELP

Windows PowerShell was developed in Redmond and was designed to ideally fit American keyboard layouts. Some commonly used keys are therefore hard to find on Swiss keyboards. I have put together a list for you here.



**Contents**

OBJECTIVES IN DRAFTING THIS BOOK .................................................................................................... 2 OTHER SOURCES OF INFORMATION ONLINE ........................................................................................... 2 USEFUL KEY COMBINATIONS FOR SWISS STANDARD KEYBOARDS ................................................................ 3 FIRST IMPRESSIONS OF WINDOWS POWERSHELL .................................................................................... 5 ADDITIONAL USES FOR OUTPUT: “PIPING” OBJECTS ................................................................................. 7 INITIAL EXERCISES WITH WINDOWS POWERSHELL OBJECTS ...................................................................... 8 WORKING WITH PROCESSES ................................................................................................................ 8 OUTPUT IN A TXT, CSV OR XML FILE ................................................................................................... 9 OUTPUT IN COLOR ............................................................................................................................. 9 CHECKING CONDITIONS USING THE IF CMDLET ......................................................................................11 OUTPUT IN HTML ..........................................................................................................................13 WORKING WITH FILES ......................................................................................................................15 FINDING OBJECT INFORMATION USING GET-MEMBER ............................................................................16 DELETING FILES ...............................................................................................................................17 CREATING FOLDERS .........................................................................................................................18 IF YOU HAVE TIME ... ........................................................................................................................21 WINDOWS POWERSHELL AS GENERIC OBJECT PROCESSING MACHINE .......................................................22 WMI OBJECTS ................................................................................................................................22 WORKING WITH .NET OBJECTS AND XML ...........................................................................................24 WORKING WITH COM OBJECTS .........................................................................................................25 WORKING WITH EVENT LOGS ............................................................................................................28 SOLUTION SCRIPTS TO THE EXERCISES IN THIS BOOK ...............................................................................29 WINDOWS POWERSHELL EXAMPLES – FROM SIMPLE TO COMPLEX ..........................................................32 THEORETICAL PRINCIPLES FOR WINDOWS POWERSHELL ........................................................................34 WINDOWS POWERSHELL – A BRIEF INTRODUCTION .............................................................................34 AIMS OF DEVELOPING WINDOWS POWERSHELL ...................................................................................34 ON TEXTS, PARSERS AND OBJECTS ......................................................................................................34 A NEW SCRIPTING LANGUAGE ............................................................................................................35 WINDOWS COMMANDS AND SERVICE PROGRAMS ................................................................................36 AN INTERACTIVE ENVIRONMENT ........................................................................................................36 SCRIPT SUPPORT .............................................................................................................................36 CMD, WSCRIPT OR POWERSHELL? DO I HAVE TO DECIDE? ...................................................................36 WINDOWS POWERSHELL 1.0 ............................................................................................................36 SECURITY WHEN USING SCRIPTS .........................................................................................................38

4

W

i

n

d

o

w

s

P

o

w

e

r

S

h

e

l

l

|

2

0

.

0

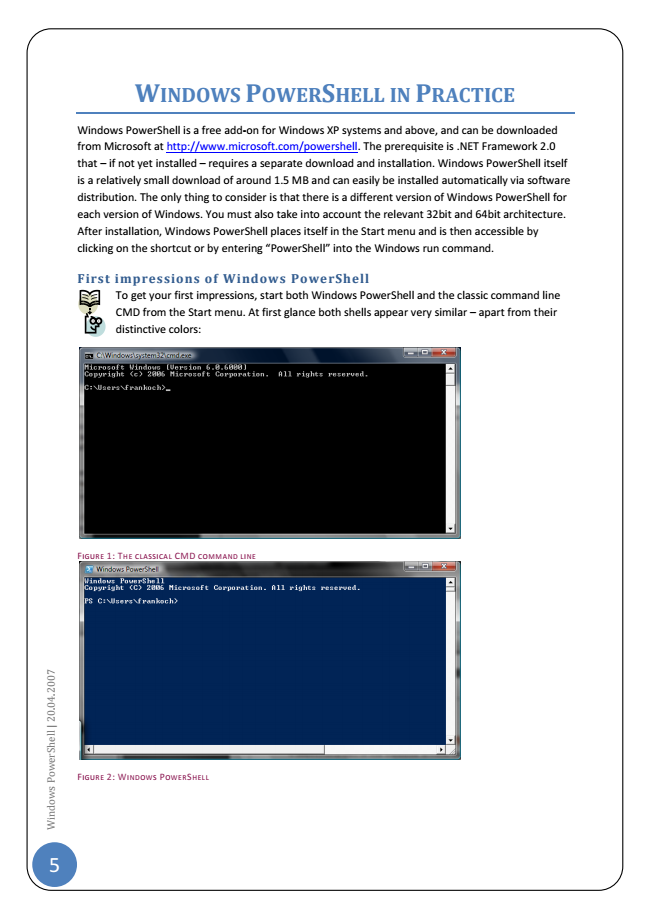
4

.2

0

0

7



5

W

i

n

d

o

w

s

P

o

w

e

r

S

h

e

l

l

|

2

0

.

0

4

.2

0

0

7

**P**

**RACTICE**

Windows PowerShell is a free add-on for Windows XP systems and above, and can be downloaded from Microsoft at http://www.microsoft.com/powershell. The prerequisite is .NET Framework 2.0 that – if not yet installed – requires a separate download and installation. Windows PowerShell itself is a relatively small download of around 1.5 MB and can easily be installed automatically via software distribution. The only thing to consider is that there is a different version of Windows PowerShell for each version of Windows. You must also take into account the relevant 32bit and 64bit architecture. After installation, Windows PowerShell places itself in the Start menu and is then accessible by clicking on the shortcut or by entering “PowerShell” into the Windows run command.

**First impressions of Windows PowerShell**

To get your first impressions, start both Windows PowerShell and the classic command line CMD from the Start menu. At first glance both shells appear very similar – apart from their distinctive colors:

F

IGURE

1: T

HE CLASSICAL

CMD

COMMAND LINE

F

IGURE

2: W

INDOWS

P

OWER

S

HELL

**W**

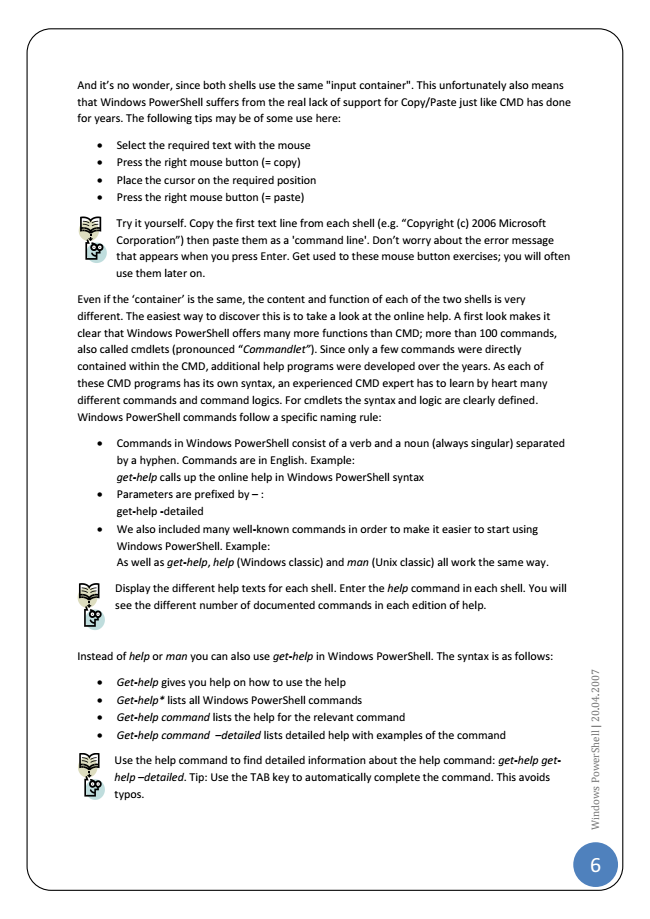
**INDOWS**

**P**

**OWER**

**S**

**HELL IN**

****

And it’s no wonder, since both shells use the same "input container". This unfortunately also means that Windows PowerShell suffers from the real lack of support for Copy/Paste just like CMD has done for years. The following tips may be of some use here:

• Select the required text with the mouse

• Press the right mouse button (= copy)

• Place the cursor on the required position

• Press the right mouse button (= paste)

Try it yourself. Copy the first text line from each shell (e.g. “Copyright (c) 2006 Microsoft Corporation”) then paste them as a 'command line'. Don’t worry about the error message that appears when you press Enter. Get used to these mouse button exercises; you will often use them later on.

Even if the ‘container’ is the same, the content and function of each of the two shells is very different. The easiest way to discover this is to take a look at the online help. A first look makes it clear that Windows PowerShell offers many more functions than CMD; more than 100 commands, also called cmdlets (pronounced “Commandlet”). Since only a few commands were directly contained within the CMD, additional help programs were developed over the years. As each of these CMD programs has its own syntax, an experienced CMD expert has to learn by heart many different commands and command logics. For cmdlets the syntax and logic are clearly defined. Windows PowerShell commands follow a specific naming rule:

• Commands in Windows PowerShell consist of a verb and a noun (always singular) separated by a hyphen. Commands are in English. Example: get-help calls up the online help in Windows PowerShell syntax

• Parameters are prefixed by – : get-help -detailed

• We also included many well-known commands in order to make it easier to start using Windows PowerShell. Example: As well as get-help, help (Windows classic) and man (Unix classic) all work the same way.

Display the different help texts for each shell. Enter the help command in each shell. You will see the different number of documented commands in each edition of help.

Instead of help or man you can also use get-help in Windows PowerShell. The syntax is as follows:

• Get-help gives you help on how to use the help

• Get-help\* lists all Windows PowerShell commands

• Get-help command lists the help for the relevant command

• Get-help command –detailed lists detailed help with examples of the command

Use the help command to find detailed information about the help command: get-help get- help –detailed. Tip: Use the TAB key to automatically complete the command. This avoids typos.

6

W

i

n

d

o

w

s

P

o

w

e

r

S

h

e

l

l

|

2

0

.

0

4

.2

0

0

7



7

W

i

n

d

o

w

s

P

o

w

e

r

S

h

e

l

l

|

2

0

.

0

4

.2

0

0

7

Additional uses for output: “piping” objects As mentioned before, Windows PowerShell is an object-oriented shell. That means that the input and output for commands are usually objects. Since humans cannot read objects, Windows PowerShell ‘translates’ the objects for output on the screen in text (professionals can even find commands in the Windows PowerShell help that enable them to adjust the output to fit their requirements). Linked commands are represented by the ‘pipe’ command: |

You can use this link to create your own Windows PowerShell book: Get-help \* | get-help – detailed does this for you: get-help \* creates a list of known commands that we use to input the command get-help –detailed. The output is extensive; you can cancel using CTRL-C.

To be able to use the results of a “help book” at a later point, it is best to move the output into a file instead of displaying it on-screen. Windows PowerShell has its own command Out-File, better known as the symbol >.

Create your own ‘book file’ now; input the following command: Get-help \* | get-help – detailed | out-file c:\Powershell-Help.txt or even get-help \* | get-help –detailed > c:\PowerShell-Help.txt. Note that you must have write authorization for the destination path (here: c:\ ).

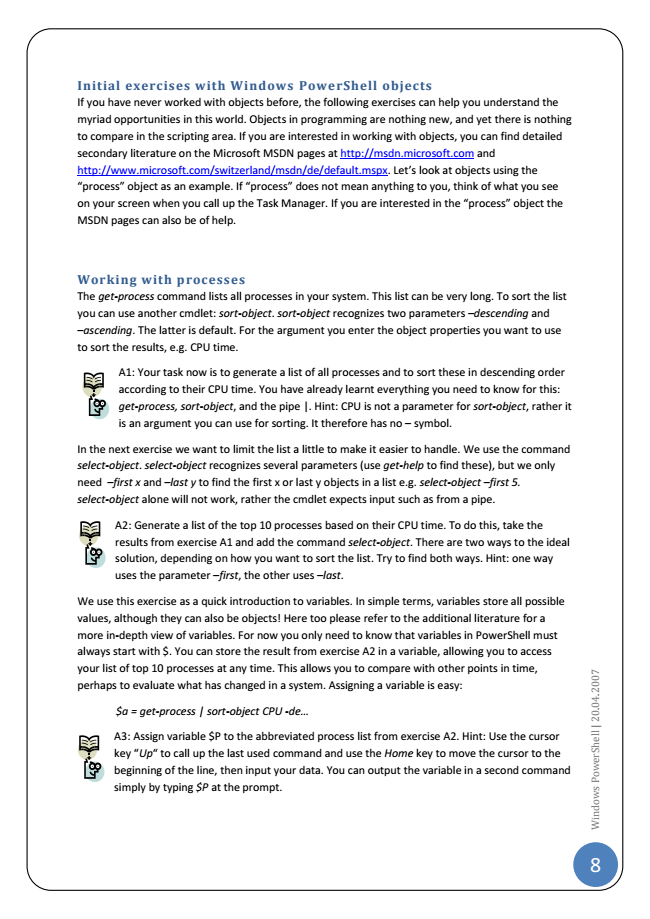
Then open your first help file in Notepad and use this as online help for the later exercises.

If you are ever searching for a command then get-help can also help you here. If you want to sort something, try to find something suitable using get-help sort\*. Get-help now starts to search for a relevant command in the Windows PowerShell command repository. Since all commands start with a verb we can structure the search easily using get-help“relevant English verb”\*. If you don’t know already, the \* symbol expresses a wildcard search, meaning that anything can come after the actual search text, we just don’t know yet and want to find everything that starts with our search text.

Once you have displayed a command (here it would be sort-object), simply call up get-help again, but this time with the relevant command and the parameter –detailed in order to find examples of how to use the command:

*get-help sort-object –detailed.*

You should now be able to solve your problem.



Initial exercises with Windows PowerShell objects If you have never worked with objects before, the following exercises can help you understand the myriad opportunities in this world. Objects in programming are nothing new, and yet there is nothing to compare in the scripting area. If you are interested in working with objects, you can find detailed secondary literature on the Microsoft MSDN pages at http://msdn.microsoft.com and http://www.microsoft.com/switzerland/msdn/de/default.mspx. Let’s look at objects using the “process” object as an example. If “process” does not mean anything to you, think of what you see on your screen when you call up the Task Manager. If you are interested in the “process” object the MSDN pages can also be of help.

Working with processes The get-process command lists all processes in your system. This list can be very long. To sort the list you can use another cmdlet: sort-object. sort-object recognizes two parameters –descending and –ascending. The latter is default. For the argument you enter the object properties you want to use to sort the results, e.g. CPU time.

A1: Your task now is to generate a list of all processes and to sort these in descending order according to their CPU time. You have already learnt everything you need to know for this: get-process, sort-object, and the pipe |. Hint: CPU is not a parameter for sort-object, rather it is an argument you can use for sorting. It therefore has no – symbol.

In the next exercise we want to limit the list a little to make it easier to handle. We use the command select-object. select-object recognizes several parameters (use get-help to find these), but we only need –first x and –last y to find the first x or last y objects in a list e.g. select-object –first 5. select-object alone will not work, rather the cmdlet expects input such as from a pipe.

A2: Generate a list of the top 10 processes based on their CPU time. To do this, take the results from exercise A1 and add the command select-object. There are two ways to the ideal solution, depending on how you want to sort the list. Try to find both ways. Hint: one way uses the parameter –first, the other uses –last.

We use this exercise as a quick introduction to variables. In simple terms, variables store all possible values, although they can also be objects! Here too please refer to the additional literature for a more in-depth view of variables. For now you only need to know that variables in PowerShell must always start with $. You can store the result from exercise A2 in a variable, allowing you to access your list of top 10 processes at any time. This allows you to compare with other points in time, perhaps to evaluate what has changed in a system. Assigning a variable is easy:

*$a = get-process | sort-object CPU -de...*

A3: Assign variable $P to the abbreviated process list from exercise A2. Hint: Use the cursor key “Up“ to call up the last used command and use the Home key to move the cursor to the beginning of the line, then input your data. You can output the variable in a second command simply by typing $P at the prompt.

8

W

i

n

d

o

w

s

P

o

w

e

r

S

h

e

l

l

|

2

0

.

0

4

.2

0

0

7



9

W

i

n

d

o

w

s

P

o

w

e

r

S

h

e

l

l

|

2

0

.

0

4

.2

0

0

7

$P F

IGURE

Output in a TXT, CSV or XML file As default, Windows PowerShell displays the results of a command chain on-screen. Any objects are converted into text so that humans can read them. We use the out-host command here. However, because Windows PowerShell wants to be efficient, this is added automatically and invisibly if you do not add it yourself. There are alternatives to out-host; find them using get-help out\*.

Outputting results as a text file is quick and easy: out-file filename is the solution. Many other shells use the > command, which is also supported by Windows PowerShell. As well as output as a text file you can also convert to a CSV or XML file. As with out-host there are separate cmdlets that perform this task for you. They are called export-CSV and export-CliXML; both require the file name as the argument. And yes, you’re right: if you can export, then you can import. Use import-CSV or import- CliXML to import the files again for viewing.

A4: Take the variable $P from exercise A3 and save its contents in a text file called “A4.txt”. Then save the contents of $P in a CSV file called “A4.CSV“, and finally again in an XML file “A4.XML“. Hint: the command > directly replaces the pipe |, which is only required for true cmdlets such as out-file, export-CSV etc. Look at the results; Notepad is sufficient for this.

Output in color Sometimes you want to highlight results to make them easier to read. You can do this, for example, by using color. The write-host command recognizes several parameters such as –foregroundcolor and –backgroundcolor. What do you think, what might be the output for the following?

*write-host “Red on blue“ –foregroundcolor red –backgroundcolor blue*

You’ve guessed it. get-help write-host –detailed gives you a list of possible colors. There are also predefined combinations: with write-warning “error“ you can also attract the user’s attention. Try this now. With this command you can output all processes in color. However, it would be more attractive if we could color the list according to additional conditions. Let’s take a closer look at that. For simplicity's sake we use your PC's services instead of the processes. If you do not know what services are, please look it up, for example on the MSDN pages. Simply put, services are the things you see listed under Control Panel / Administrative Tools / Services. The nice thing about them is that

3: O

UTPUT FOR VARIABLE