



Being Lazy with Microsoft Windows Powershell

Developer Network Bristol

23rd September 2008

Good morning. Welcome to “Being Lazy with Microsoft Windows PowerShell.” My name’s Ben Lamb.

Windows PowerShell is Microsoft’s replacement for the command-prompt. It combines a command-line interface with a new scripting language and I really like it. It borrows ideas from just about every previous shell and scripting language ever invented and introduces a few of its own.

The best thing about it is that it’s written in .NET and fairly easy to extend so you can use it for your own purposes. This talk is going to show you 3 ways you can reuse PowerShell rather than reinventing the wheel.

It’s not a PowerShell tutorial and assumes no previous knowledge. I’ll be explaining the relevant PowerShell concepts as we go.

Can I have a show of hands for people who have used PowerShell before? Who’s used the SDK to develop cmdlets?

All You Need to Know

- Most commands are Cmdlets
- Cmdlets produce objects
- A **pipeline** allows you to pass **objects** between cmdlets

In PowerShell the commands are known as cmdlets. They return objects. PowerShell has the concept of a pipeline, you can pass the output of one cmdlet into another cmdlet and so on.

This is similar to the pipe operator in MS-DOS and Unix shells with one crucial difference. You're transferring objects not text. Thus you can easily access individual properties on the objects and cmdlets can adjust their behaviour depending on the type of objects they receive.

Demo 1 – PowerShell Basics

- Example of cmdlets and pipelines

This PowerShell, it looks just like the MS DOS prompt.

Run “dir”. Even dir works!

If I type “ps” I get a list of running processes. Since there is only one command in the pipeline the results are display on screen.

I can pass the output to another command to filter the list of objects. Finally I can pass the filtered list of objects to the kill command which kills the processes.

Demo 1 - Recap

- Basic PowerShell cmdlets
- Aliases
- Pipelines
- Calling static .NET methods

Scenarios

- Writing Cmdlets
- Hosting PowerShell

In this talk I'm going to cover 3 scenarios.

Writing your own cmdlets - this is the core of extending PowerShell.

PowerShell can be customised and turned into a dedicated tool for managing a particular application. I believe this is what Microsoft do with the latest version of Exchange. We'll look at how to create your own custom shell.

Finally you can incorporate PowerShell into your own applications and allow them to be scripted.

Creating Cmdlets

- Create a PowerShell "SnapIn"
 - These contain code + meta data + other stuff

Demo 2 - PowerFlickr

- Flickr is an on-line photo sharing tool from Yahoo
- Allows users to upload photos, group them into photosets and add tags and comments
- (And a lot more besides)
- Provides an API which someone as wrapped as part of the Flickr.Net project

PowerShell cmdlets can be written in any .NET language. Lots of third parties have extended PowerShell by writing cmdlets, some very useful ones are included in PowerShell Community Extensions which is freely downloadable from CodePlex. Developers can save a lot of time by using a command-line interface and even more time by using a scripting language to automate repetitive tasks. That's why we use msbuild to handle our deployments rather than copying the files manually with Windows Explorer don't we?

I've worked on business applications in the past where we wanted a quick management interface for support purposes so I wanted to show you how to create cmdlets to do this in PowerShell.

Unfortunately I don't have a suitable business application I can show you so I'm going to use Flickr instead.

Flickr is a photo sharing web based application operated by Yahoo. It allows users to upload photos, share them, add comments and tags and many other things. It provides an API for developers.

We're lazy so rather than deal with API directly we're going to make use of a library that Joe Bloggs has uploaded to Codeplex that wraps it nicely into .NET.

What PowerShell Provides

- Parsing of the command line
- Validation of arguments
- Support for “WhatIf” and “confirm”

Demo 3 Hosting PowerShell

Very often you want extensibility in your application.

Being Lazy with PowerShell 2

- Remote Execution
- Background Jobs
- Ability to Write Cmdlets in PowerShell
- Graphical Interface

At the beginning of May Microsoft released a second Community Technology Preview of PowerShell 2. It is available for public download but you should read all the warnings before trying it.

It adds some interesting new features.

Takeaways

- Lazy programmers don't reinvent the wheel
- Adding management interfaces to your enterprise applications is a good idea
- PowerShell can be used to provide scripting support for your apps
- Learning how to use PowerShell might make you more productive

Lazy programmers don't reinvent the wheel.