Working with Data in PowerShell

EXPORTING, IMPORTING, AND CONVERTING DATA

Matt Allford

@mattallford www.mattallford.com

Prerequisites

Working with PowerShell console

Running and piping commands

Using PowerShell parameters

PowerShell language familiarity

Course Scenario

Dan Stanton

Globomantics desktop support engineer

Anna Fisher

Globomantics engineering manager

Steve Matthews

Globomantics developer

Course Overview

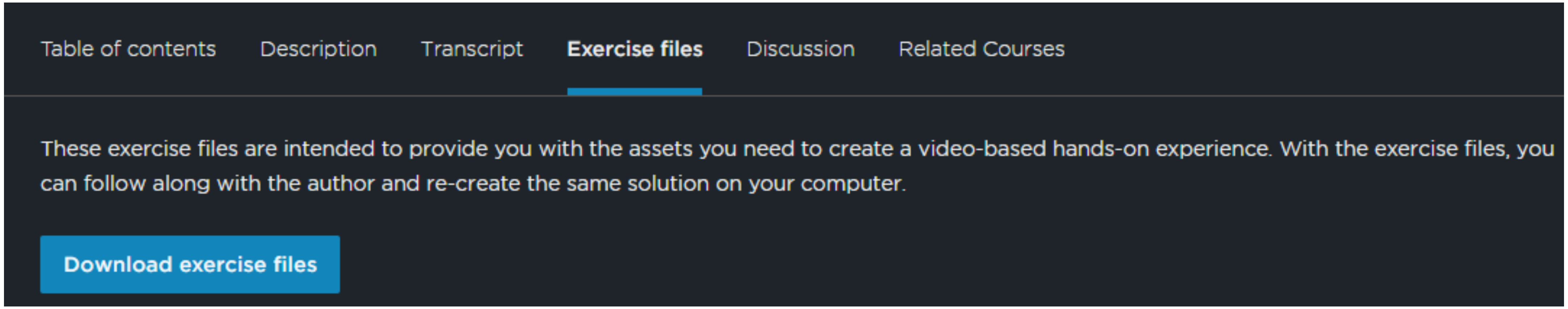
Exporting, importing, and converting data

- Comma Separated Value (CSV)
- eXtensible Markup Language (XML)
- JavaScript Object Notation (JSON)
- Hyper Text Markup Language (HTML)

Getting data from the internet

Exercise Files and Demo Environment

Exercise Files



These exercise files are intended to provide you with the assets you need to create a video-based hands-on experience. With the exercise files, you can follow along with the author and re-create the same solution on your computer.

Comma Separated Values

Comma delimited text file

Each line is a record

Each record contains one or more fields

Typically has a heading

- But isn’t mandatory

`Firstname,Lastname,City,Icecream`	◀ Heading
`Mary,Jacobson,Melbourne,Blueberry`	◀ Record
`Derek,Lakeson,Brisbane,Rainbow`	◀ Record
`Ashlee,Sands,Tokyo,Chocolate`	◀ Record
`Jess,Davis,London,Strawberry`	◀ Record
`Zoe,Fields,,Mint`	◀ Record

CSV Benefits and Drawbacks

Benefits	Drawbacks
Easily read by a human	Less versatile than other data structures
Easy to edit	Doesn’t support a hierarchy and nested objects
Well established and widely supported	Values with commas or carriage returns need additional work
Compact and light weight	You have to establish what fields are, usually with a heading row

Export-Csv

Takes PowerShell objects and exports to a CSV file

PowerShell object properties become CSV headings

Useful parameters:

- InputObject
- Append
- IncludeTypeInformation (PS 6.0)
- Delimiter

Import-Csv

Creates custom objects from items in a CSV file

CSV Heading becomes property name

- Parameter “-Header” can alter the behavior

Comma delimiter by default, can be changed

Property values are converted to strings

Converting CSV

ConvertTo-Csv

ConvertFrom-Csv

Demo

Working with CSV in PowerShell

Extensible Markup Language (XML)

Markup language

Designed to store and transport data

- Just information wrapped in tags

Good choice for a hierarchy

```xml
<?xml version="1.0" encoding="UTF-8"?>
<IcecreamData>
  <person>
    <Firstname>Mary</Firstname>
    <Lastname>Jacobson</Lastname>
    <City>Melbourne</City>
    <Icecream>Blueberry</Icecream>
  </person>
  <person>
    <Firstname>Derek</Firstname>
    <Lastname>Lakeson</Lastname>
    <City>Brisbane</City>
    <Icecream>Rainbow</Icecream>
  </person>
</IcecreamData>
```

◀ Prolog

◀ Root Element

◀ Child Element

◀ Child Element

XML Benefits and Drawbacks

Benefits	Drawbacks
Based on international standards	Verbose
Widely supported, platform and language independent	Typically large in size
Allows storing and transport of hierarchical data	Less readable compared to other data formats (JSON, YAML, etc.)
Supports Unicode	

PowerShell and XML

Export-CliXml

Creates an XML-based representation of an object, or objects, and stores it in a file

Import-CliXml

Imports a CLIXML file and creates corresponding objects in PowerShell

ConvertTo-Xml

Creates an XML-based representation of an object

Demo

Working with XML in PowerShell

JavaScript Object Notation (JSON)

Uses JavaScript syntax, but is text only

Language independent

Lightweight data format, commonly used for sending and receiving data with REST

```
[
    {
        "Firstname": "Mary",
        "Lastname": "Jacobson",
        "City": "Melbourne",
        "Icecream": "Blueberry"
    }
]
```

- Square brackets hold arrays
- Curly braces hold objects
- Data is in name / value pairs

```
[
    {
        "Firstname": "Mary",
        "Lastname": "Jacobson",
        "City": "Melbourne",
        "Icecream": [
            "Blueberry",
            "Banana"
        ]
    }
]
```

- Square brackets hold arrays
- Curly braces hold objects
- Data is in name / value pairs

```
[
    {
        "Firstname": "Mary",
        "Lastname": "Jacobson",
        "City": "Melbourne",
        "Icecream": [
            "Blueberry",
            "Banana"
        ]
    },
    {
        "Firstname": "Derek",
        "Lastname": "Lakeson",
        "City": "Brisbane",
        "Icecream": "Rainbow"
    }
]
```

- Square brackets hold arrays
- Curly braces hold objects
- Data is in name / value pairs

JSON Benefits and Drawbacks

Benefits	Drawbacks
Lightweight format for storing and transporting data	Can be limited in terms of supported data types
“Self-describing” and easy to understand	Can’t use comments
Smaller and fast, easy to parse	Can be difficult for humans to read
Less verbose than XML, provides structure CSV doesn’t	

PowerShell and JSON

ConvertFrom-Json

Converts a JSON-Formatted string to a custom object or hash table

ConvertTo-Json

Converts an object to a JSON-formatted string

Demo

Working with JSON in PowerShell

Hyper Text Markup Language (HTML)

The standard markup language for creating web pages

HTML describes the structure of the web page

- Doesn’t contain structured data

Styling performed with Cascading Style Sheets (CSS)

PowerShell usage with ConvertTo-Html

```html
<!DOCTYPE html>

<html>

<head>

<title>My Icecream Report</title>

</head>

<body>


<h1>Icecream Report</h1>

<p>This report contains information
about people and their favorite
icecream.</p>


</body>

</html>
```

- ◀ Doctype
- ◀ Root Element
- ◀ Contains meta information about the page
- ◀ Shown in the browser
- ◀ Defines the document’s body
- ◀ Defines a large heading
- ◀ Defines a paragraph

Demo

Working with HTML in PowerShell

Summary

Choose the best data structure

Importing, exporting, and converting

Experiment exporting / converting the same PowerShell object to different data types

Leveraging HTML can produce great reports

Up Next:
Getting Data from the Internet