PowerShell Serialization

TRAINSIGNAL

THE GLOBAL LEADER IN PROFESSIONAL COMPUTER TRAINING

What is Serialization?

- Objects are rich data sources
  - Methods
  - Properties
  - PowerShell customizations
- Serialization "saves" the object to a file
  - CSV
  - XML
- Serialized objects can be recreated from these sources
  - Even if they were created elsewhere

Serialization Scenario

You're troubleshooting a server performance problem and need help

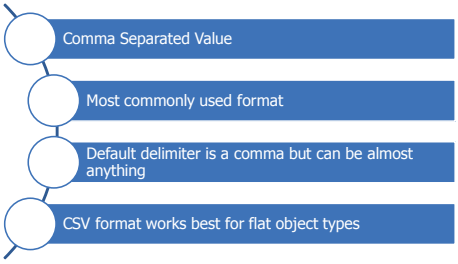
You export processes to an XML file (serialization)

Send file to senior administrator

…who imports the file on her machine (deserialization)

She now sees all of the processes just as they were when exported

CSV



Export-CSV

Turn objects into a CSV file	– Specify file name – Default delimiter is comma but you can change it
Properties become CSV headings	– Be careful of what properties you convert! – Arrays and objects don't export
Includes a TypeInformation comment	– Used on import to recreate the objects in PowerShell – Can be omitted if using CSV outside of PowerShell
Append parameter is new in 3.0	– Make sure you export same properties

Import-CSV

- PowerShell can import any CSV file
- Default delimiter is comma but you can specify
- Headings become property names
  - You can specify an alternate header
  - Tip: Align header names with cmdlet parameter names
- All values are strings
- No methods are defined
- Import with TypeInformation creates a CSV:Selected object

Converting CSV

Objects can be converted to and from CSV without a file

Useful to modify CSV data "on the fly"

ConvertTo-CSV and ConvertFrom-CSV

- Same basic parameters and Import and Export cmdlets
- Save to a variable
- `PS C:\> $csvdata = dir c:\work | select fullname,length,lastwritetime | convertto-csv`

Data can still be saved to a file

```
PS C:\> $csvData | out-file c:\work\data.csv
```


CSV Demonstrations

XML

- PowerShell supports the full .NET XML library
- XML best choice for complex or rich objects
- Captured in a hierarchical structure
- PowerShell's XML serialization targeted for use within PowerShell

Export-Clixml

Serialize objects to XML

- Specify file name
- Specify serialization depth
- No append feature

Intended to be used with Import-Clixml

Object type information captured

```
PS C:\> dir c:\work | export-clixml
work.xml
```


Import-Clixml

Deserialize content from XML files created with Export-Clixml

Object types are deserialized

Property types are maintained

No object methods created on import

```
PS C:\> $files = import-clixml work.xml
```


ConvertTo-XML

Serialize objects to a standard XML document

- Use in non-PowerShell applications
- Cannot be imported using Import-Clixml

Save XML to a variable

XML can be modified in place

Invoke the Save() method to write to a file

Working with XML documents is beyond the scope of this course

```
PS C:\> $xml = get-eventlog -list | convertto-xml
PS C:\> $xml.Save("c:\work\evlog.xml")
```

XML Demonstrations

Serialization Summary

Know your data

- Flat = CSV
- Rich = XML

How will your serialized files be used?

- In PowerShell?
- In an external application?

No methods are serialized

Align property names with cmdlets

LAB

1. Export all running services to a CSV file, capturing the service name, display name, status, and whether the service can be stopped using the colon (:) as the delimiter.
2. Import the csv file and display the objects sorted by display name
3. Export the newest 100 errors from the system eventlog to a PowerShell xml file.
4. Import the xml file and group the results by the event source.