Name Click here to enter name ID Click here to enter id

Data presented in XML

Exercise 4.2

# Instructions

Answer all questions directly in this document. You will save and upload this completed document as your homework submission.

# Overview

One of the most ubiquitous formats for data presentation is *Extensible Markup Language* (XML). It forms the basis of a lot of data delivery protocols, including:

* Rich Syndication Summary (RSS), used for news and podcast feeds;
* Simple Object Access Protocol (SOAP), used for automated web transactions;
* Security Assertion Markup Language (SAML), used to regulate access-control in web services;

and many others. A great reference for XML syntax reference is at:   
<https://www.w3schools.com/xml/xml_syntax.asp> . Please browse and skim through that article before you begin this exercise. (It should only require a minute or two; it’s pretty short.)

In PowerShell, the infrastructure to leverage XML data is provided by .Net’s **System.Xml** class. PowerShell exposes each node of the tree structure of every XML document as its own **XmlElement** object, so you can take any branch of an XML tree and work with it independently. Furthermore, whenever an XML node is a collection (array) of similar objects, you will see in the exercises below that PowerShell easily works with entire collections, even deep into subtree structures of elements of a collection.

# Requirements

* Internet connection
* psfiles.zip

# Setup

If you haven’t already, download and extract the contents of **psfiles.zip**. (It has a custom XML file you will use in tasks 3 through 6.)

# Task 1—Working with a syndication feed

## Steps

1. The SANS Technology Institute produces *Internet Storm Center’s Daily Stormcast*, a brief security news podcast. Download the current list of its available episodes. Windows:   
   **Invoke-WebRequest https://isc.sans.edu/dailypodcast.xml -OutFile C:\TEMP\p.xml**   
   (Linux or macOS: adjust the output file’s temporary folder to **/tmp** instead of **C:\TEMP**.)
2. Have a look at it:   
   **notepad C:\TEMP\p.xml** # (Again, adjust to use your favorite text viewer in Linux/macOS)
   1. Notice the first “angle-bracketed tag” in the file:   
      **<?xml version="1.0" encoding="UTF-8"?>**   
      You should recognize this tag as the standard XML *prologue*.
   2. Notice that the next tag begins with **rss**. This is the *root element* of this XML document. Scroll all the way down to the bottom; the last line is its corresponding closing tag **</rss>**. An XML node can have optional *attributes*. The first attribute of the **rss** node is named **version** and has the value **2.0**.
   3. *Optional challenge*: as you examine the RSS data, see if you can figure out what kinds of information are presented between each **<item>** (opening) and **</item>** (closing) paired tags.
   4. *Optional challenge*: use <https://acronymfinder.com/>, search engines, or a generative AI chat tool to learn various meanings for what RSS stands for.
3. Use PowerShell to parse the RSS feed’s XML data:   
   **$x = [xml] (Get-Content C:\TEMP\p.xml)**   
   (*Note*: if this throws an error, then the XML file is not formatted correctly; try to download it again.)
4. What class of object is in **$x**?   
   **$x.GetType().Name**   
   Your output: Click or tap here to enter text.
   1. Verify the name of the document’s root element:   
      **$x | Get-Member -MemberType Properties**
   2. What class of object is the root element?   
      **$x.rss.GetType().Name**   
      Your output Click or tap here to enter text.
   3. View the values of various sub-elements inside the root element:   
      **$x.rss**
   4. View values of various sub-elements inside the **channel** sub-element:   
      **$x.rss.channel**
   5. The item elements you explored in step 2 were not shown. The .NET library treated them differently. Enter:   
      **$x.rss.channel | Get-Member**   
      What is the **MemberType** of **Item**? Click or tap here to enter text.
   6. Query **Item**’s type:   
      **$x.rss.channel.Item.GetType().Name**   
      Your output: Click or tap here to enter text. (This means that the Item node in the XML structure is not just one branch of the tree. It’s a *collection*.)
   7. View values of the item sub-elements:   
      **$x.rss.channel.item**   
      View them in table-format instead of list-format:   
      **$x.rss.channel.item | ft**   
      Is it easier to view them in table format? What’s your opinion? Click or tap here to enter text.
5. Now let’s focus on the most recent episode’s data:   
   **$x.rss.channel.item[0]**
   1. What is the most recent episode’s number? Click or tap here to enter text.
   2. When was its publication date? Click or tap here to enter text.
   3. What type of file is contained in its enclosure? Click or tap here to enter text. (Hints: look at the filename extension at the end of the **guid** node’s data, and try looking at the **url** and **type** *attributes* of the **enclosure** node.)
   4. How long (in minutes and seconds) is the episode? Click or tap here to enter text.
6. It seems more convenient to listen to the episode in your favorite podcast app, but just for this exercise, let’s try to download and listen to it with PowerShell!   
   *Windows:*   
   **Invoke-WebRequest $x.rss.channel.item[0].guid -OutFile C:\TEMP\p.mp3**   
   **start C:\TEMP\p.mp3**   
   *macOS:*   
   **Invoke-WebRequest $x.rss.channel.item[0].guid -OutFile /tmp/p.mp3**   
   **open /tmp/p.mp3**   
   *Linux:*   
   (I don’t know, but I bet you can figure it out! The Windows **start** and macOS **open** commands shown above will play the downloaded file using whatever app is registered as the default MP3 player. Linux distributions vary. Some distros might not pre-install an MP3 player, so you might need to search for and install a player app.)

# Task 2—Working with SAML data

For this task, please use Microsoft Windows, so that we can use its security certificate viewer in the last step.

## Steps

1. Change to your temporary folder:   
   **Set-Location C:\TEMP**
2. Create a new empty file, which will soon hold a SAML identity provider document:   
   **New-Item idp.xml**
3. Browse to this page of Oracle’s documentation web site:   
   <https://docs.oracle.com/en/cloud/saas/field-service/faadu/c-sampleMetadataXML.html#Sample-Metadata-XML-File-for-SAML-Identity-Provider>   
   Copy *all* of the XML data in the sample file shown there, from the prologue line  
   <?xml version="1.0"?>   
   all the way through the closing root entity tag  
   </md:EntityDescriptor>.
4. Use Notepad to edit your new **idp.xml** file:   
   **notepad idp.xml**   
   Paste the contents you copied from the Oracle documentation web page. Save the file, then exit (close) your Notepad editor.
5. Now parse that XML document:   
   **$x = [xml] (Get-Content idp.xml)**
   1. Look at your new XML object:  
      **$x**   
      What is the name of the XML document’s root element? Click or tap here to enter text.
6. Start “climbing down” through its tree structure:   
   **$x.EntityDescriptor**   
   **$x.EntityDescriptor.IDPSSODescriptor**   
   **$x.EntityDescriptor.IDPSSODescriptor.KeyDescriptor**
   1. At this point, look at the **use** property. What two *uses* are supported by this identity provider? Click or tap here to enter text.
   2. Since there is more than one use, we know we have encountered another collection! For convenience, save this node of the XML tree in a new variable:   
      **$kd = $x.EntityDescriptor.IDPSSODescriptor.KeyDescriptor**
   3. Keep climbing deeper into the tree structure. Notice that PowerShell still follows the whole collection as you go further through the subtree nodes:   
      **$kd.KeyInfo**   
      **$kd.KeyInfo.X509Data**   
      **$kd.KeyInfo.X509Data.X509Certificate**
7. To extract the signing certificate and save it in a file named **idps.cer**, enter:  
   **$kd.KeyInfo.X509Data.X509Certificate[0] > idps.cer**   
   To extract the encryption certificate and save it in a file named **idpc.cer**, enter:   
   **$kd.KeyInfo.X509Data.X509Certificate[1] > idpc.cer**
8. Launch the Windows Certificate app, to decode and view the data in the signing certificate:   
   **start idps.cer**   
    *Warning*: *do NOT tap the* [Install certificate…] *button!*  
   Are the ***Issued to*** and ***Issued by*** values identical? Click or tap here to enter text. (*If they are the same, then this is a self-signed certificate, which by default would not be trusted*.)
9. Launch the Certificate app again, this time to decode and view the data in the encryption certificate:   
   **start idpc.cer**   
   Then tap the [Details] tab.
   1. What is the value of the *Thumbprint* field? Click or tap here to enter text.
   2. Is the Thumbprint value the same in both certificates? Click or tap here to enter text. (*If they are identical, then this identity provider uses the same certificate for both uses*.)

# Task 3—Read a Custom XML File and View Its Contents

For the remaining tasks, the XML data are organized as follows. Note that this custom XML document does NOT contain a prologue tag. Indeed, for any XML document, the prologue line is optional.

<lapidary>

<metals>

<metal>

<Symbol>Ag</Symbol>

<Name>Silver (fine)</Name>

<MeltingPoint>1762</MeltingPoint>

<SpecificGravity>10.6</SpecificGravity>

</metal>

<metal>

<Symbol>Al</Symbol>

<Name>Aluminum</Name>

<MeltingPoint>1220</MeltingPoint>

<SpecificGravity>2.7</SpecificGravity>

</metal>

</metals>

<gems>

<gem>

<GID>2</GID>

<Mineral>Almandite</Mineral>

<Hardness>7.5</Hardness>

<RefractiveIndex>1.8</RefractiveIndex>

<CrownAngle>37</CrownAngle>

<PavilionAngle>42</PavilionAngle>

<CriticalAngle>34</CriticalAngle>

<Common>false</Common>

</gem>

<gem>

<GID>3</GID>

<Mineral>Anatase</Mineral>

<Hardness>5.75</Hardness>

<RefractiveIndex>2.524</RefractiveIndex>

<CrownAngle>35</CrownAngle>

<PavilionAngle>41</PavilionAngle>

<CriticalAngle>24</CriticalAngle>

<Common>false</Common>

</gem>

</gems>

</lapidary>

## Steps

1. Change your current directory to the **psfiles/data** subdirectory.
2. Convert the XML data in lapidary.xml into an XML object:   
   $x=[xml](Get-Content .\lapidary.xml)
3. Explore the XML document’s root node with PowerShell.  
   $x
   1. You should see the root node of the document, which in this case is **lapidary**.
4. Explore one node farther into the XML tree. Enter:   
   $x.lapidary
   1. You should see two nodes, **metals** and **gems**. Let’s climb down into the metals node first.
5. Enter  
   $x.lapiary.metals
   1. You should recognize a collection of **metal** objects.
6. To see the entire collection of metal objects, enter:   
   $x.lapidary.metals.metal
   1. You should see many **metal** objects with several properties. List **metal**’s four property names: Click or tap here to enter text.
7. Reformat the output as list or table (whichever is different from what you observed in step 6):   
   $x.lapidary.metals.metal|fl or $x.lapidary.metals.metal|ft

# Task 4—Process Individual Collections from a custom XML data file

You can work collections easily. Let’s use a foreach loop to iterate through a collection.

## Steps

1. The melting point temperature data in lapidary.xml are measured in Farenheight. Let’s convert them to Celsius. Enter (on one line):   
   foreach($m in $x.lapidary.metals.metal) {"$($m.name) melts at $(($m.meltingpoint-32) /1.8)C"}
   1. What is the melting point for titanium, in both Fahrenheit and Celsius? Click or tap here to enter text.
2. For convenience, work with subsets of data by assigning a node in the XML document to a variable. Create a variable that contains just the metals:   
   $metals=$x.lapidary.metals.metal
   1. View the results  
      $metals
3. Do the same for gems  
   $gems=$x.lapidary.gems.gem
4. Sort the metals by Name:   
   $metals|sort name
   1. What is the melting point of the first metal in the alphabetized list? Click or tap here to enter text.
5. Sort by specific gravity. Which is the heaviest metal? Click or tap here to enter text.

# Task 5—Modifying XML data

## Steps

1. *Change nodes* - Modify the XML data to update the melting point temperatures to Celsius to the nearest degree. After the conversion we will cast the result to an [int], to round the data to the nearest degree. However, an XML node can only contain strings, so we will need to convert the rounded results back to strings. For this task, you will use the **$metals** variable from the previous task. (It contains just the metal nodes.) Enter (all on one line):   
   foreach($m in $metals) {$m.meltingpoint=[string][int](($m.meltingpoint-32) /1.8)}
   1. Did this command alter the data in the original XML object (**$x**)? Click or tap here to enter text.
2. *Add node* - Let’s add a new element to the metals collection. We’ll add *Unobtainium*, and document that it melts at *5000* degrees and has a specific gravity of *.75*.
   1. First, we need a new Metal node:   
      $nm=$x.CreateElement('Metal')
   2. Next, create elements for each property of the metal, and set the value for each property:   
      $symbol=$x.CreateElement('Symbol')  
      $symbol.InnerText='Uo'  
      $Name=$x.CreateElement('Name')  
      $Name.InnerText='Unobtainium'  
      $mp=$x.CreateElement('MeltingPoint')  
      $mp.InnerText=5000  
      $sg=$x.CreateElement('SpecificGravity')  
      $sg.InnerText=.75
   3. Then attach each property element as a “child” to the new metal element  
      $nm.AppendChild($symbol)  
      $nm.AppendChild($name)  
      $nm.AppendChild($mp)  
      $nm.AppendChild($sg)
      1. View our new metal object:   
         $nm
   4. Add the new metal to the Metals node.  
      $x.lapidary.metals.AppendChild($nm)
   5. Now view your new xml data  
      $x.lapidary.metals.metal
      1. Is Unobtainium there? Click or tap here to enter text.
   6. *Remove nodes* - Let’s remove the gems from this XML document.  
      $x.lapidary.gems.RemoveAll()
      1. View the contents of the xml document:   
         $x.lapidary   
         Are the gems gone? Click or tap here to enter text.

# Task 6—Save the Changes

## Steps

1. Look at the members of our XML object:   
   **$x | get-member**
2. Notice there is a Save method. Let’s save our modified XML data. The XML object is unaware of PowerShell’s current directory, so we will need to construct a path to save the file. Enter:   
   **$filename=Join-Path (Get-Location) 'MetalsInC.xml'**   
   $x.Save($filename)
3. View the contents of your new file:   
   Get-Content $filename  
   Does it have Unobtainium, no gems, and all its melting point temperatures are in Celsius? Click or tap here to enter text.

# Deliverable

Upload this document with completed answers to I-Learn Canvas.